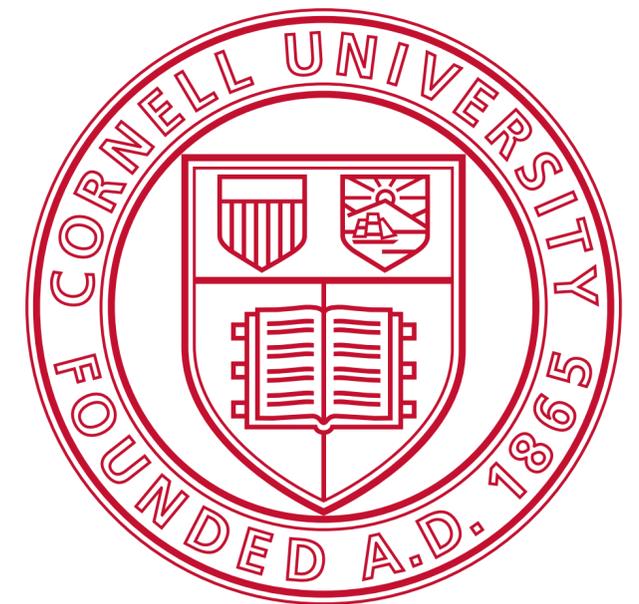


LLMs for Databases: the Next Five Years

Immanuel Trummer



How do You Know a Human Wrote This?

The New York Times, 2020

Meet GPT-3. It Has Learned to Code (and Blog and Argue).

The New York Times, 2020

An A.I. bot answers 10 burning questions about the 2020 NFL season

USA Today, 2020

The Jessica Simulation: Love and loss in the age of A.I.

The San Francisco Chronicle, 2021

Bringing People Back to Life With the Power of AI Chatbots.

Forbes, 2021

Meet GPT-3, the natural-language system that generates tweets, pens poetry, summarizes emails, answers trivia, translates languages and even writes its own computer programs

Chicago Tribune, 2021

'Grassroots' bot campaigns are coming. Governments don't have a plan to stop them.

Washington Post, 2021

ChatGPT Could be AI's iPhone Moment

Bloomberg, 2022

The New Chatbots Could Change the World.

The New York Times, 2022

ChatGPT Gained One Million Users in Under a Week. Here's Why the Chatbot is Primed to Disrupt Search as We Know It.

Fortune, 2022

ChatGPT and How AI Disrupts Industries.

Harvard Business Review, 2022



What's on your mind today?

Ask anything

+  Tools





Agent K v1.0

Large Language Models Orchestrating Structured Reasoning Achieve Kaggle Grandmaster Level

Antoine Grosnit^{1,3,†}, Alexandre Maraval^{1,†}, James Doran¹, Giuseppe Paolo¹, Albert Thomas¹, Refinath Shahul Hameed Nabeezath Beevi¹, Jonas Gonzalez¹, Khyati Khandewal¹, Ignacio Iacobacci¹, Abdelhakim Benechehab¹, Hamza Cherkaoui¹, Youssef Attia El-Hilli¹, Kun Shao¹, Jianye Hao¹, Jun Yao¹

Can Foundation Models Wrangle Your Data?

Avnika Narayan Ines Chami Laurel Orr Christopher Ré
 Stanford University Numbers Station Stanford University Stanford University
 avnikn@cs.stanford.edu ines.chami@numbersstation.ai lorrl@cs.stanford.edu chrisre@cs.stanford.edu

ABSTRACT
 Foundation Models (FMs) are models trained on large corpora of data that, at very large scale, can generalize to new tasks without any task-specific finetuning. As these models continue to grow in

OpenSearch-SQL: Enhancing Text-to-SQL with Dynamic Few-shot and Consistency Alignment

XIANGJIN XIE, Alibaba Cloud Computing, China
 GUANGWEI XU, Alibaba Cloud Computing, China
 LINGYAN ZHAO, Alibaba Cloud Computing, China
 RUIJIE GUO, Alibaba Cloud Computing, China

Although multi-agent collaborative Large Language Models (LLMs) have achieved significant breakthroughs in the Text-to-SQL task, their performance is still constrained by various factors. These factors include the incompleteness of the framework, failure to follow instructions, and model hallucinations. To address these problems, we propose OpenSearch-SQL, which divides the Text-to-SQL task into four main modules: Preprocessing, Extraction, Generation, and Refinement, along with an Alignment module based on a consistency alignment mechanism. This architecture aligns the inputs and outputs of agents through the Alignment module, reducing failures in instruction following and hallucination. Furthermore, we introduce SQL-Like (an intermediate language), optimize the structured Chain-of-Thought (CoT) based on SQL-Like, and develop a dynamic few-shot strategy via self-taught Query-CoT-SQL.

In terms of model selection, we directly applied the base LLMs without any post-training, thereby simplifying the task chain and enhancing the framework's portability. Experimental results show that OpenSearch-SQL achieves an execution accuracy (EX) of 69.3% on the BIRD development set, 72.28% on the test set, and a reward-based validity efficiency score (R-VES) of 69.36%, with all three metrics ranking first at the time of submission. These results demonstrate the comprehensive advantages of the proposed method in both effectiveness and efficiency.

CCS Concepts: • **Computing methodologies** → *Multi-agent systems*; • **Information systems** → **Structured Query Language**; • **Theory of computation** → *Parsing*.

Additional Key Words and Phrases: Text-to-SQL, Multi-Agent, Retrieval

ACM Reference Format:
 Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. 2025. OpenSearch-SQL: Enhancing Text-to-SQL with Dynamic Few-shot and Consistency Alignment. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 194 (June 2025), 24 pages. <https://doi.org/10.1145/3725331>

1 Introduction

Text-to-SQL task aims to automatically generate Structured Query Language (SQL) queries from Natural Language Queries (NLQ). This task can improve the ability to access databases without the need for knowledge of SQL [17].

As the text-to-SQL problem is notoriously hard, these systems have been a long-standing challenge of the database community for several decades [17]. Early work defined query answers

Authors' Contact Information: Xiangjin Xie, xie.xiangjin.xxj@alibaba-inc.com, Alibaba Cloud Computing, Hangzhou, China; Guangwei Xu, guangwei.xu@alibaba-inc.com, Alibaba Cloud Computing, Hangzhou, China; Lingyan Zhao, zhaolingyan.zly@alibaba-inc.com, Alibaba Cloud Computing, Hangzhou, China; Ruijie Guo, ruijie.guo@alibaba-inc.com, Alibaba Cloud Computing, Beijing, China.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
 ACM 2636-4573/2025/6-ART194
<https://doi.org/10.1145/3725331>

Proc. ACM Manag. Data, Vol. 3, No. 3 (SIGMOD), Article 194. Publication date: June 2025.



Agent K v1.0

Large Language Models Orchestrating Structured Reasoning Achieve Kaggle Grandmaster Level

Antoine Grosnit^{1,3,†}, Alexandre Maraval^{1,†}, James Doran¹, Giuseppe Paolo¹, Albert Thomas¹, Refinath Shahul Hameed Nabeezath Beevi¹, Jonas Gonzalez¹, Khyati Khandewal¹, Ignacio Iacobacci¹, Abdelhakim Benechehab¹, Hamza Cherkaoui¹, Youssef Attia El-Hilli¹, Kun Shao¹, Jianye Hao¹, Jun Yao¹

Can Foundation Models Wrangle Your Data?

Avanika Narayan
Stanford University
avanika@cs.stanford.edu

Ines Chami
Numbers Station
ines.chami@numbersstation.ai

Laurel Orr
Stanford University
lorr1@cs.stanford.edu

Christopher Ré
Stanford University
chrisre@cs.stanford.edu

ABSTRACT
Foundation Models (FM) are models trained on large corpora of data that, at very large scale, can generalize to new tasks without any task-specific finetuning. As these models continue to grow in

OpenSearch-SQL: Enhancing Text-to-SQL with Dynamic Few-shot and Consistency Alignment

XIANGJIN XIE, Alibaba Cloud Computing, China
GUANGWEI XU, Alibaba Cloud Computing, China
LINGYAN ZHAO, Alibaba Cloud Computing, China
RUIJIE GUO, Alibaba Cloud Computing, China

Although multi-agent collaborative Large Language Models (LLMs) have achieved significant breakthroughs in the Text-to-SQL task, their performance is still constrained by various factors. These factors include the incompleteness of the framework, failure to follow instructions, and model hallucinations. To address these problems, we propose OpenSearch-SQL, which divides the Text-to-SQL task into four main modules: Preprocessing, Extraction, Generation, and Refinement, along with an Alignment module based on a consistency alignment mechanism. This architecture aligns the inputs and outputs of agents through the Alignment module, reducing failures in instruction following and hallucination. Furthermore, we introduce SQL-Like (an intermediate language), optimize the structured Chain-of-Thought based on SQL-Like, and develop a dynamic few-shot strategy via self-taught Query-CoT-SQL.

In terms of model selection, we directly applied the base LLMs without any post-training, thereby simplifying the task chain and enhancing the framework's portability. Experimental results show that OpenSearch-SQL achieves an execution accuracy (EX) of 69.3% on the BIRD development set, 72.28% on the test set, and a reward-based validity efficiency score (R-VES) of 69.36%, with all three metrics ranking first at the time of submission. These results demonstrate the comprehensive advantages of the proposed method in both effectiveness and efficiency.

CCS Concepts • **Computing methodologies** → *Multi-agent systems*; • **Information systems** → **Structured Query Language**; • **Theory of computation** → *Parsing*.

Additional Key Words and Phrases: Text-to-SQL, Multi-Agent, Retrieval

ACM Reference Format:
Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. 2025. OpenSearch-SQL: Enhancing Text-to-SQL with Dynamic Few-shot and Consistency Alignment. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 194 (June 2025), 24 pages. <https://doi.org/10.1145/3725331>

1 Introduction

Text-to-SQL task aims to automatically generate Structured Query Language (SQL) queries from Natural Language Queries (NLQ). This task can improve the ability to access databases without the need for knowledge of SQL [17].

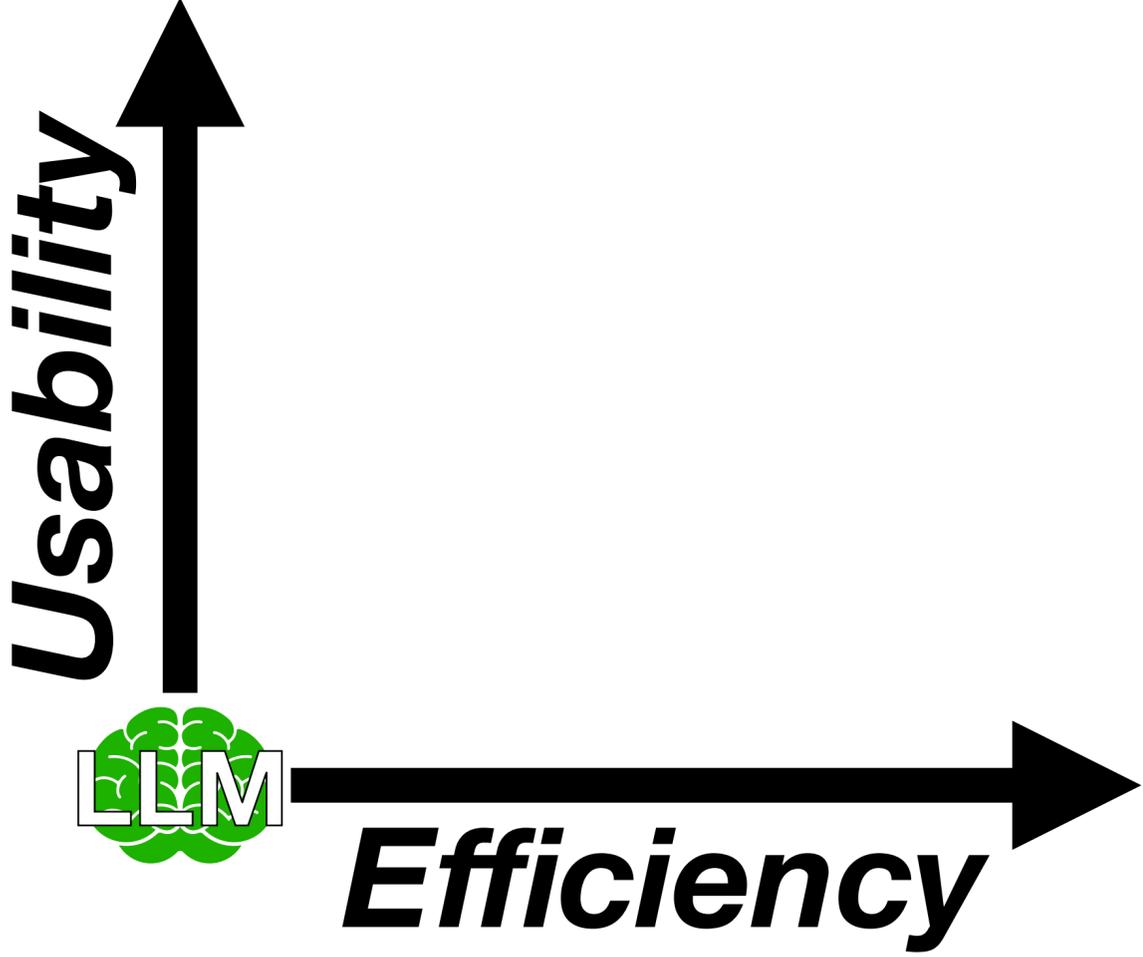
As the text-to-SQL problem is notoriously hard, these systems have been a long-standing challenge of the database community for several decades [17]. Early work defined query answers

Authors' Contact Information: Xiangjin Xie, xiexiangjin.xxj@alibaba-inc.com, Alibaba Cloud Computing, Hangzhou, China; Guangwei Xu, guangwei.xu@alibaba-inc.com, Alibaba Cloud Computing, Hangzhou, China; Lingyan Zhao, zhaolingyan.zly@alibaba-inc.com, Alibaba Cloud Computing, Hangzhou, China; Ruijie Guo, ruijie.guo@alibaba-inc.com, Alibaba Cloud Computing, Beijing, China.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 2836-6575/2025/6-ART194
<https://doi.org/10.1145/3725331>

Proc. ACM Manag. Data, Vol. 3, No. 3 (SIGMOD), Article 194. Publication date: June 2025.



Demonstrating SQLBarber: Leveraging Large Language Models to Generate Customized and Realistic SQL Workloads

Jiale Lao
Cornell University
lhaca, New York, USA
jlao@cs.cornell.edu

Immanuel Trummer
Cornell University
Ithaca, New York, USA
itrummer@cornell.edu

Abstract 1 Introduction

Query Rewriting via Large Language Models

Jie Liu
University of Michigan
jiezliu@umich.edu

Barran Mozafari
University of Michigan
mozafari@umich.edu

ABSTRACT
Query rewriting is one of the most effective techniques for coping with slowly writing queries before issuing them down to the query engines. **Challenge of manual query rewriting. The costly system** query to begin with, Query rewriting, whether done manually or automatically, has remained a challenging task.

λ-Tune: Harnessing Large Language Models for Automated Database System Tuning

VICTOR GIANNAKOURIS, Cornell University, USA
IMMANUEL TRUMMER, Cornell University, USA

We introduce λ-Tune, a framework that leverages Large Language Models (LLMs) for automated database system tuning. The design of λ-Tune is motivated by the capabilities of the latest generation of LLMs. Different from prior work, leveraging LLMs to extract tuning hints for single parameters, λ-Tune generates entire configuration scripts, based on a large input document, describing the tuning context. λ-Tune generates alternative configurations, using a principled approach to identify the best configuration, out of a small set of candidates. In doing so, it minimizes reconfiguration overheads and ensures that evaluation costs are bounded as a function of the optimal run time. By treating prompt generation as a cost-based optimization problem, λ-Tune conveys the most relevant context to the LLM while bounding the number of input tokens and, therefore, monetary fees for LLM invocations. We compare λ-Tune to various baselines, using multiple benchmarks and PostgreSQL and MySQL as target systems for tuning, showing that λ-Tune is significantly more robust than prior approaches.

CCS Concepts • **Information systems** → **Autonomous database administration**; **Database utilities and tools**.

Additional Key Words and Phrases: Database, Tuning, Large Language Models

ACM Reference Format:
Victor Giannakouris and Immanuel Trummer. 2025. λ-Tune: Harnessing Large Language Models for Automated Database System Tuning. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 2 (February 2025), 26 pages. <https://doi.org/10.1145/3709652>

1 Introduction

The performance of database management system changes dramatically as a function of various tuning choices, including settings for system configuration parameters as well as physical design choices such as indexing, sorting, or partitioning. This has motivated a large body of research on automated database system tuning. Recent work exploits machine learning to find near-optimal configurations [5, 6, 14, 24] but suffers from high training and exploration overheads. This has motivated a new line of research [12, 21], exploiting LLMs to heuristically prune the search space for tuning. Similar to human database administrators, such models leverage commonsense knowledge, extracted from text documents, to narrow the focus to tuning options that seem "reasonable", given the tuning context. This paper presents λ-Tune [7] (Language Models for Better Database Administration), a system that exploits capabilities offered by the latest generation of LLMs, including the likes of GPT-4 and Claude 3, to optimize various tuning choices for specific systems and OLAP workloads, including system parameter settings as well as physical design decisions.

λ-Tune. Prior approaches to LLM-enhanced database tuning [12, 21] parse text documents (e.g., Authors' Contact Information: Victor Giannakouris, Cornell University, Ithaca, NY, USA, vg292@cornell.edu; Immanuel Trummer, Cornell University, Ithaca, NY, USA, it224@cornell.edu.

This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2025 Copyright held by the owner/author(s).
ACM 2836-6575/2025/2-ART2
<https://doi.org/10.1145/3709652>

Proc. ACM Manag. Data, Vol. 3, No. 1 (SIGMOD), Article 2. Publication date: February 2025.

Agent K v1.0

Large Language Models Orchestrating Structured Reasoning Achieve Kaggle Grandmaster Level

Antoine Grosni^{1,3,†}, Alexandre Maraval^{1,3,†}, James Doran¹, Giuseppe Paolo¹, Albert Thomas¹, Refinath Shahul Hameed Nabeezath Beevi¹, Jonas Gonzalez¹, Khyati Khandewal¹, Ignacio Iacobacci¹, Abdelhakim Benechehab¹, Hamza Cherkaoui¹, Youssef Attia El-Hilli¹, Kun Shao¹, Jianye Hao¹, Jun Yao¹

Can Foundation Models Wrangle Your Data?

Avanika Narayan
Stanford University
avanika@cs.stanford.edu

Ines Chami
Numbers Station
ines.chami@numbersstation.ai

Laurel Orr
Stanford University
lorr1@cs.stanford.edu

Christopher Ré
Stanford University
chrisre@cs.stanford.edu

ABSTRACT
Foundation Models (FMs) are models trained on large corpora of data that, at very large scale, can generalize to new tasks without any task-specific finetuning. As these models continue to grow in

No
Foundation Model

OpenSearch-SQL: Enhancing Text-to-SQL with Dynamic Few-shot and Consistency Alignment

XIANGJIN XIE, Alibaba Cloud Computing, China
GUANGWEI XU, Alibaba Cloud Computing, China
LINGYAN ZHAO, Alibaba Cloud Computing, China
RUIJIE GUO, Alibaba Cloud Computing, China

Although multi-agent collaborative Large Language Models (LLMs) have achieved significant breakthroughs in the Text-to-SQL task, their performance is still constrained by various factors. These factors include the incompleteness of the framework, failure to follow instructions, and model hallucinations. To address these problems, we propose OpenSearch-SQL, which divides the Text-to-SQL task into four main modules: Preprocessing, Extraction, Generation, and Refinement, along with an Alignment module based on a consistency alignment mechanism. This architecture aligns the inputs and outputs of agents through the Alignment module, reducing failures in instruction following and hallucination. Furthermore, we introduce SQL-Like (an intermediate language), optimize the structured Chain-of-Thought based on SQL-Like, and develop a dynamic few-shot strategy via self-taught Query-CoT-SQL.

In terms of model selection, we directly applied the base LLMs without any post-training, thereby simplifying the task chain and enhancing the framework's portability. Experimental results show that OpenSearch-SQL achieves an execution accuracy (EX) of 69.3% on the BIRD development set, 72.28% on the test set, and a reward-based validity efficiency score (R-VES) of 69.36%, with all three metrics ranking first at the time of submission. These results demonstrate the comprehensive advantages of the proposed method in both effectiveness and efficiency.

CCS Concepts • **Computing methodologies** → *Multi-agent systems*; • **Information systems** → **Structured Query Language**; • **Theory of computation** → *Parsing*.

Additional Key Words and Phrases: Text-to-SQL, Multi-Agent, Retrieval

ACM Reference Format:
Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. 2025. OpenSearch-SQL: Enhancing Text-to-SQL with Dynamic Few-shot and Consistency Alignment. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 194 (June 2025), 24 pages. <https://doi.org/10.1145/3725331>

1 Introduction

Text-to-SQL task aims to automatically generate Structured Query Language (SQL) queries from Natural Language Queries (NLQ). This task can improve the ability to access databases without the need for knowledge of SQL [17].

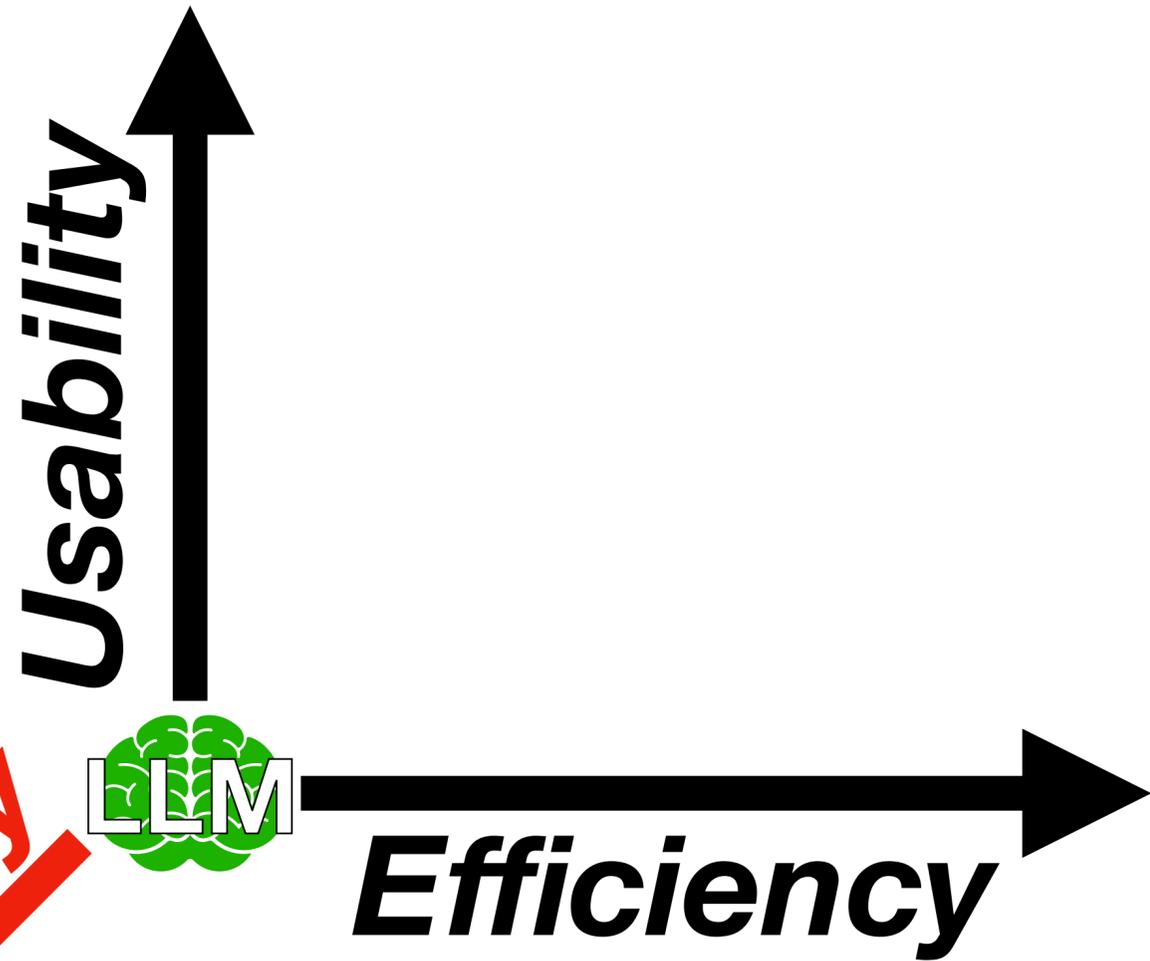
As the text-to-SQL problem is notoriously hard, these systems have been a long-standing challenge of the database community for several decades [17]. Early work defined query answers

Authors' Contact Information: Xiangjin Xie, xiexiangjin.xxj@alibaba-inc.com, Alibaba Cloud Computing, Hangzhou, China; Guangwei Xu, guangwei.xu@alibaba-inc.com, Alibaba Cloud Computing, Hangzhou, China; Lingyan Zhao, zhaolingyan.zly@alibaba-inc.com, Alibaba Cloud Computing, Hangzhou, China; Ruijie Guo, ruijie.guo@alibaba-inc.com, Alibaba Cloud Computing, Beijing, China.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 2836-6575/2025/6-ART194
<https://doi.org/10.1145/3725331>

Proc. ACM Manag. Data, Vol. 3, No. 3 (SIGMOD), Article 194. Publication date: June 2025.



Demonstrating SQLBarber: Leveraging Large Language Models to Generate Customized and Realistic SQL Workloads

Jiale Lao
Cornell University
lhaca, New York, USA
jlao@cs.cornell.edu

Immanuel Trummer
Cornell University
Ithaca, New York, USA
itrummer@cornell.edu

Abstract 1 Introduction

Query Rewriting via Large Language Models

Jie Liu
University of Michigan
jiezliu@umich.edu

Barran Mozafari
University of Michigan
mozafari@umich.edu

ABSTRACT
Query rewriting is one of the most effective techniques for coping with slowly writing queries before issuing them down to the query engines. However, it has remained a challenging task. The possible system query to begin with, Query rewriting, whether done manually or automatically, has remained a challenging task. The possible system

J-Tune: Harnessing Large Language Models for Automated Database System Tuning

VICTOR GIANNAKOURIS, Cornell University, USA
IMMANUEL TRUMMER, Cornell University, USA

We introduce J-Tune, a framework that leverages Large Language Models (LLMs) for automated database system tuning. The design of J-Tune is motivated by the capabilities of the latest generation of LLMs. Different from prior work, leveraging LLMs to extract tuning hints for single parameters, J-Tune generates entire configuration scripts, based on a large input document, describing the tuning context. J-Tune generates alternative configurations, using a principled approach to identify the best configuration, out of a small set of candidates. In doing so, it minimizes reconfiguration overheads and ensures that evaluation costs are bounded as a function of the optimal run time. By treating prompt generation as a cost-based optimization problem, J-Tune conveys the most relevant context to the LLM while bounding the number of input tokens and, therefore, monetary fees for LLM invocations. We compare J-Tune to various baselines, using multiple benchmarks and PostgreSQL and MySQL as target systems for tuning, showing that J-Tune is significantly more robust than prior approaches.

CCS Concepts • **Information systems** → *Autonomous database administration*; **Database utilities and tools**.

Additional Key Words and Phrases: Database, Tuning, Large Language Models

ACM Reference Format:
Victor Giannakouris and Immanuel Trummer. 2025. J-Tune: Harnessing Large Language Models for Automated Database System Tuning. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 2 (February 2025), 26 pages. <https://doi.org/10.1145/3709652>

1 Introduction

The performance of database management system changes dramatically as a function of various tuning choices, including settings for system configuration parameters as well as physical design choices such as indexing, sorting, or partitioning. This has motivated a large body of research on automated database system tuning. Recent work exploits machine learning to find near-optimal configurations [5, 6, 14, 24] but suffers from high training and exploration overheads. This has motivated a new line of research [12, 21], exploiting LLMs to heuristically prune the search space for tuning. Similar to human database administrators, such models leverage commonsense knowledge, extracted from text documents, to narrow the focus to tuning options that seem "reasonable", given the tuning context. This paper presents J-Tune [7] (Language Models for Better Database Administration), a system that exploits capabilities offered by the latest generation of LLMs, including the likes of GPT-4 and Claude 3, to optimize various tuning choices for specific systems and OLAP workloads, including system parameter settings as well as physical design decisions.

J-Tune. Prior approaches to LLM-enhanced database tuning [12, 21] parse text documents (e.g., Authors' Contact Information: Victor Giannakouris, Cornell University, Ithaca, NY, USA, vg292@cornell.edu; Immanuel Trummer, Cornell University, Ithaca, NY, USA, it224@cornell.edu).

This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2025 Copyright held by the owner/author(s).
ACM 2836-6575/2025/2-ART2
<https://doi.org/10.1145/3709652>

Proc. ACM Manag. Data, Vol. 3, No. 1 (SIGMOD), Article 2. Publication date: February 2025.



CrowdDB: Answering Queries with Crowdsourcing

Michael J. Franklin
AMPLab, UC Berkeley
franklin@cs.berkeley.edu

Donald Kossmann
Systems Group, ETH Zurich
donaldk@inf.ethz.ch

Tim Kraska
AMPLab, UC Berkeley
kraska@cs.berkeley.edu

Sukriti Ramesh
Systems Group, ETH Zurich
ramesh@student.ethz.ch

Reynold Xin
AMPLab, UC Berkeley
rxin@cs.berkeley.edu

ABSTRACT

Some queries cannot be answered by machines only. Processing such queries requires human input for providing information that is missing from the database, for performing computationally difficult functions, and for matching, ranking, or aggregating results based on fuzzy criteria. CrowdDB uses human input via crowdsourcing to process queries that neither database systems nor search engines can adequately answer. It uses SQL both as a language for posing complex queries and as a way to model data. While CrowdDB leverages many aspects of traditional database systems, there are also important differences. Conceptually, a major change is that the traditional closed-world assumption for query processing does not hold for human input. From an implementation perspective, human-oriented query operators are needed to solicit, integrate and cleanse crowdsourced data. Furthermore, performance and cost depend on a number of new factors including worker affinity, training, fatigue, motivation and location. We describe the design of CrowdDB, report on an initial set of experiments using Amazon Mechanical Turk, and outline important avenues for future work in the development of crowdsourced query processing systems.

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems

General Terms

Human Factors, Languages, Design, Performance

1. INTRODUCTION

Relational database systems have achieved widespread adoption, not only in the business environments for which they were originally envisioned, but also for many other types of structured data, such as personal, social, and even scientific information. Still, as data creation and use become increasingly democratized through web, mobile and other technologies, the limitations of the technology are becoming more apparent. RDBMSs make several key

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD '11, June 12–16, 2011, Athens, Greece.
Copyright 2011 ACM 978-1-4503-0661-4/11/06...\$10.00.

assumptions about the correctness, completeness and unambiguity of the data they store. When these assumptions fail to hold, relational systems will return incorrect or incomplete answers to user questions, if they return any answers at all.

1.1 Power to the People

One obvious situation where existing systems produce wrong answers is when they are missing information required for answering the question. For example, the query:

```
SELECT market_capitalization FROM company
WHERE name = "I.B.M.";
```

will return an empty answer if the company table instance in the database at that time does not contain a record for "I.B.M.". Of course, in reality, there are many reasons why such a record may be missing. For example, a data entry mistake may have omitted the I.B.M. record or the record may have been inadvertently deleted. Another possibility is that the record was entered incorrectly, say, as "I.B.N."

Traditional systems can erroneously return an empty result even when no errors are made. For example, if the record was entered correctly, but the name used was "International Business Machines" rather than "I.B.M." This latter "entity resolution" problem is not due to an error but is simply an artifact of having multiple ways to refer to the same real-world entity.

There are two fundamental problems at work here. First, relational database systems are based on the "Closed World Assumption": information that is not in the database is considered to be false or non-existent. Second, relational databases are extremely literal. They expect that data has been properly cleaned and validated before entry and do not natively tolerate inconsistencies in data or queries.

As another example, consider a query to find the best among a collection of images to use in a motivational slide show:

```
SELECT image FROM picture
WHERE topic = "Business Success"
ORDER BY relevance LIMIT 1;
```

In this case, unless the relevance of pictures to specific topics has been previously obtained and stored, there is simply no good way to ask this question of a standard RDBMS. The issue here is one of judgement: one cannot reliably answer the question simply by applying relational operators on the database.

All of the above queries, however, while unanswerable by today's relational database systems, could easily be answered by people, especially people who have access to the Internet.

Crowdsourced Databases: Query Processing with People

Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, Robert C. Miller
MIT CSAIL
(marcu, srrice, karger, madden, rcm)@csail.mit.edu

ABSTRACT

Amazon's Mechanical Turk ("MTurk") service allows users to post short tasks ("HITS") that other users can receive a small amount of money for completing. Common tasks on the system include labelling a collection of images, combining two sets of images to identify people which appear in both, or extracting sentiment from a corpus of text snippets. Designing a workflow of various kinds of HITS for filtering, aggregating, sorting, and joining data sources together is common, and comes with a set of challenges in optimizing the cost per HIT, the overall time to task completion, and the accuracy of MTurk results. We propose Quirk, a novel query system for managing these workflows, allowing crowd-powered processing of relational databases. We describe a number of query execution and optimization challenges, and discuss some potential solutions.

1. INTRODUCTION

Amazon's Mechanical Turk service (<https://www.mturk.com/mturk/welcome>) ("MTurk") allows users to post short tasks ("HITS") that other users ("turkers") can receive a small amount of money for completing. A HIT creator specifies how much he or she will pay for a completed task. Example HITS involve compiling some information from the web, labeling the subject of an image, or comparing two documents. More complicated tasks, such as ranking a set of ten items or completing a survey are also possible. MTurk is used widely to perform data analysis tasks which are either easier to express to humans than to computers, or for which there aren't yet effective artificial intelligence algorithms.

Task prices vary from a few cents (\$.01-\$.02/HIT is a common price) to several dollars for completing a survey. MTurk has around 100,000-300,000 HITS posted at any time (<http://mturk-tracker.com/general/>). Novel uses of MTurk include matching earthquake survivor pictures with missing persons in Haiti (<http://app.beextra.org/mission/show/missionid/605/mode/do>), authoring a picture book (<http://bjjoern.org/projects/catbook/>), and embedding turkers as editors into a word processing system [2].

From the point of view of a database system, crowd-powered tasks can be seen as operators—similar to user-defined functions—that are invoked as a part of query pro-

This article is published under a Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0/>), which permits distribution and reproduction in any medium as well allowing derivative works, provided that you attribute the original work to the author(s) and CIDR 2011. CIDR 2011

cessing. For example, given a database storing a table of images, a user might want to query for images of flowers, generating a HIT per image to have turkers perform the filter task. Several challenges arise in processing such a workflow. First, each HIT can take minutes to return, requiring an asynchronous query executor. Second, in addition to considering time, a crowdworker-aware optimizer must consider monetary cost and result accuracy. Finally, the selectivity of operators can not be predicted *a priori*, requiring an adaptive approach to query processing.

In this paper, we propose Quirk, a crowdworker-aware database system which addresses these challenges. Quirk can issue HITS that extract, order, filter, and join complex datatypes, such as images and large text blobs. While we describe Quirk here using the language of MTurk (turkers and HITS), and our initial prototype runs on MTurk, we aim for Quirk to be crowd-platform-independent. Future versions of Quirk will compile tasks for different kinds of crowds with different interfaces and incentive systems. Quirk is a new system in active development; we describe our vision for the system, propose a high level sketch of a UDF-like syntax for executing these HITS, and explore a number of questions that arise in such a system, including:

- What is the HIT equivalent of various relational operations? For example, an equijoin HIT might require humans to identify equal items, whereas a HIT-based sort can use human comparators.
- How many HITS should be generated for a given task? For example, when sorting, one can generate a HIT that asks users to score many items, or to simply compare two. How can the system optimize each HIT?
- Given that a large database could result in millions of HITS, how should the system choose which HITS to issue? Given that only a fraction of items in a large database can have HITS generated for them, what is the proper notion of query correctness?
- How much to charge for a HIT? Higher prices can lead to faster answers. Can the system adaptively optimize the price of HITS based on user-specified response time and budget constraints?
- Who is the right crowdworker for a task? MTurk provides paid workers, but an enterprise might prefer expert workers, and an operation with a tight budget might look for non-monetary incentives.

2. MOTIVATING EXAMPLES

Here is a list of tasks that Quirk should be able to run:



Deco: Declarative Crowdsourcing

Aditya Parameswaran
Stanford University
Stanford, CA
adityagp@cs.stanford.edu

Hyunjung Park
Stanford University
Stanford, CA
hyunjung@cs.stanford.edu

Hector Garcia-Molina
Stanford University
Stanford, CA
hector@cs.stanford.edu

Neoklis Polyzotis
UC Santa Cruz
Santa Cruz, CA
alkis@cs.ucsc.edu

Jennifer Widom
Stanford University
Stanford, CA
widom@cs.stanford.edu

ABSTRACT

Crowdsourcing enables programmers to incorporate "human computation" as a building block in algorithms that cannot be fully automated, such as text analysis and image recognition. Similarly, humans can be used as a building block in data-intensive applications—providing, comparing, and verifying data used by applications. Building upon the decades-long success of declarative approaches to conventional data management, we use a similar approach for data-intensive applications that incorporate humans. Specifically, declarative queries are posed over stored relational data as well as data computed on-demand from the crowd, and the underlying system orchestrates the computation of query answers.

We present Deco, a database system for declarative crowdsourcing. We describe Deco's data model, query language, and our prototype. Deco's data model was designed to be *general* (it can be instantiated to other proposed models), *flexible* (it allows methods for data cleansing and external access to be plugged in), and *principled* (it has a precisely-defined semantics). Syntactically, Deco's query language is a simple extension to SQL. Based on Deco's data model, we define a precise semantics for arbitrary queries involving both stored data and data obtained from the crowd. We then describe the Deco query processor which uses a novel push-pull hybrid execution model to respect the Deco semantics while coping with the unique combination of latency, monetary cost, and uncertainty introduced in the crowdsourcing environment. Finally, we experimentally explore the query processing alternatives provided by Deco using our current prototype.

Categories and Subject Descriptors

H.2.1 [Database Management]: Logical Design—*data models*;
H.2.3 [Database Management]: Languages—*query languages*

Keywords

declarative crowdsourcing, human computation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CIKM '12, October 29–November 2, 2012, Maui, HI, USA.
Copyright 2012 ACM 978-1-4503-1156-4/12/10...\$15.00.

1. INTRODUCTION

Crowdsourcing [7, 26] uses human workers to capture or generate data on demand and/or to classify, rank, label or enhance existing data. Often, the tasks performed by humans are hard for a computer to do, e.g., rating a new restaurant or identifying features of interest in a video. We can view the human-generated data as *a data source*, so naturally one would like to seamlessly integrate the crowd data source with other conventional sources, so the end user can interact with a single, unified database. And naturally one would like a *declarative* system, where the end user describes the needs, and the system dynamically figures out what and how to obtain crowd data, and how it must be integrated with other data. This overall vision, and the underlying issues and challenges, were outlined in our earlier paper [23].

In this paper, we realize our earlier vision to present Deco (short for "declarative crowdsourcing"), a database system that answers declarative queries posed over stored relational data, the collective knowledge of the crowd, as well as other external data sources. Our goal is to make Deco appear to the end user as similar as possible to a conventional database system (a relational one in our case), while hiding many of the complexities of dealing with humans as data sources (e.g., breaking down large tasks into smaller ones, posting tasks to a marketplace and pricing them, dealing with latency, and handling errors and inconsistencies in human-provided data).

We describe the Deco data model, the query language and semantics, and the query processor.

While the idea of declarative access to crowd data is appealing and natural, there are significant challenges to address:

- How do we resolve disagreeing human opinions? For instance, if we collect five ratings for a movie, we may want to give the end user the average, but if we collect five phone numbers, we may want to instead eliminate duplicates. How does the schema designer (or DBA) specify what to do, and when during query execution do we resolve the opinions?
- How does the database system interact with the human workers (the crowd)? For instance, to get restaurant information, we may want Deco to give the worker the name of a restaurant (e.g., "Bouchon"), and ask for its cuisine. But in other cases Deco may want to give the worker the cuisine (e.g., "French"), and ask for restaurants serving that cuisine. Or Deco may ask for cuisine and rating given the name of a restaurant. How does the schema designer define the available options, which we can view as different "access methods"? Are there restrictions on the access methods that can be defined? And how does Deco decide what access method to use for a given query? How do we enable Deco to use



CrowdDB: Answering Queries with Crowdsourcing

Michael J. Franklin
AMPLab, UC Berkeley
franklin@cs.berkeley.edu

Donald Kossmann
Systems Group, ETH Zurich
donaldk@inf.ethz.ch

Tim Kraska
AMPLab, UC Berkeley
kraska@cs.berkeley.edu

Sukriti Ramesh
Systems Group, ETH Zurich
ramesh@student.ethz.ch

Reynold Xin
AMPLab, UC Berkeley
rxin@cs.berkeley.edu

ABSTRACT

Some queries cannot be answered by machines only. Processing such queries requires human input for providing information that is missing from the database, for performing computationally difficult functions, and for matching, ranking, or aggregating results based on fuzzy criteria. CrowdDB uses human input via crowdsourcing to process queries that neither database systems nor search engines can adequately answer. It uses SQL both as a language for posing complex queries and as a way to model data. While CrowdDB leverages many aspects of traditional database systems, there are also important differences. Conceptually, a major change is that the traditional closed-world assumption for query processing does not hold for human input. From an implementation perspective, human-oriented query operators are needed to solicit, integrate and cleanse crowdsourced data. Furthermore, performance and cost depend on a number of new factors including worker affinity, training, fatigue, motivation and location. We describe the design of CrowdDB, report on an initial set of experiments using Amazon Mechanical Turk, and outline important avenues for future work in the development of crowdsourced query processing systems.

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems

General Terms

Human Factors, Languages, Design, Performance

1. INTRODUCTION

Relational database systems have achieved widespread adoption, not only in the business environments for which they were originally envisioned, but also for many other types of structured data, such as personal, social, and even scientific information. Still, as data creation and use become increasingly democratized through web, mobile and other technologies, the limitations of the technology are becoming more apparent. RDBMSs make several key

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD '11, June 12–16, 2011, Athens, Greece.
Copyright 2011 ACM 978-1-4503-0661-4/11/06...\$10.00.

Crowdsourced Databases: Query Processing with People

Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, Robert C. Miller
MIT CSAIL
(marcu, sircice, karger, madden, rcm)@csail.mit.edu

ABSTRACT

Amazon's Mechanical Turk ("MTurk") service allows users to post short tasks ("HITs") that other users can receive a small amount of money for completing. Common tasks on the system include labelling a collection of images, combining two sets of images to identify people which appear in both, or extracting sentiment from a corpus of text snippets. Designing a workflow of various kinds of HITs for filtering, aggregating, sorting, and joining data sources together is common, and comes with a set of challenges in optimizing the cost per HIT, the overall time to task completion, and the accuracy of MTurk results. We propose Qurk, a novel query system for managing these workflows, allowing crowd-powered processing of relational databases. We describe a number of query execution and optimization challenges, and discuss some potential solutions.

1. INTRODUCTION

Amazon's Mechanical Turk service (<https://www.mturk.com/mturk/welcome>) ("MTurk") allows users to post short tasks ("HITs") that other users ("turkers") can receive a small amount of money for completing. A HIT creator specifies how much he or she will pay for a completed task. Example HITs involve compiling some information from the web, labeling the subject of an image, or comparing two documents. More complicated tasks, such as ranking a set of ten items or completing a survey are also possible. MTurk is used widely to perform data analysis tasks which are either easier to express to humans than to computers, or for which there aren't yet effective artificial intelligence algorithms. Task prices vary from a few cents (\$.01-\$0.02/HIT is a common price) to several dollars for completing a survey. MTurk has around 100,000-300,000 HITs posted at any time (<http://mturk-tracker.com/general/>). Novel uses of MTurk include matching earthquake survivor pictures with missing persons in Haiti (<http://app.boextra.org/mission/show/missionid/605/node/do>), authoring a picture book (<http://bjoern.org/projects/catbook/>), and embedding turkers as editors into a word processing system [2]. From the point of view of a database system, crowd-powered tasks can be seen as operators—similar to user-defined functions—that are invoked as a part of query pro-

This article is published under a Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0/>), which permits distribution and reproduction in any medium as well allowing derivative works, provided that you attribute the original work to the author(s) and CIDR 2011. CIDR 2011

cessing. For example, given a database storing a table of images, a user might want to query for images of flowers, generating a HIT per image to have turkers perform the filter task. Several challenges arise in processing such a workflow. First, each HIT can take minutes to return, requiring an asynchronous query executor. Second, in addition to considering time, a crowdworker-aware optimizer must consider monetary cost and result accuracy. Finally, the selectivity of operators can not be predicted *a priori*, requiring an adaptive approach to query processing.

In this paper, we propose Qurk, a crowdworker-aware database system which addresses these challenges. Qurk can issue HITs that extract, order, filter, and join complex datatypes, such as images and large text blobs. While we describe Qurk here using the language of MTurk (turkers and HITs), and our initial prototype runs on MTurk, we aim for Qurk to be crowd-platform-independent. Future versions of Qurk will compile tasks for different kinds of crowds with different interfaces and incentive systems. Qurk is a new system in active development; we describe our vision for the system, propose a high level sketch of a UDF-like syntax for executing these HITs, and explore a number of questions that arise in such a system, including:

- What is the HIT equivalent of various relational operations? For example, an equijoin HIT might require humans to identify equal items, whereas a HIT-based sort can use human comparators.
- How many HITs should be generated for a given task? For example, when sorting, one can generate a HIT that asks users to score many items, or to simply compare two. How can the system optimize each HIT?
- Given that a large database could result in millions of HITs, how should the system choose which HITs to issue? Given that only a fraction of items in a large database can have HITs generated for them, what is the proper notion of query correctness?
- How much to charge for a HIT? Higher prices can lead to faster answers. Can the system adaptively optimize the price of HITs based on user-specified response time and budget constraints?
- Who is the right crowdworker for a task? MTurk provides paid workers, but an enterprise might prefer expert workers, and an operation with a tight budget might look for non-monetary incentives.

2. MOTIVATING EXAMPLES

Here is a list of tasks that Qurk should be able to run:



Deco: Declarative Crowdsourcing

Aditya Parameswaran
Stanford University
Stanford, CA
adityagg@cs.stanford.edu

Hyunjung Park
Stanford University
Stanford, CA
hyunjung@cs.stanford.edu

Hector Garcia-Molina
Stanford University
Stanford, CA
hector@cs.stanford.edu

Neoklis Polyzotis
UC Santa Cruz
Santa Cruz, CA
alkis@cs.ucsc.edu

Jennifer Widom
Stanford University
Stanford, CA
widom@cs.stanford.edu

ABSTRACT

Crowdsourcing enables programmers to incorporate "human computation" as a building block in algorithms that cannot be fully automated, such as text analysis and image recognition. Similarly, humans can be used as a building block in data-intensive applications—providing, comparing, and verifying data used by applications. Building upon the decades-long success of declarative approaches to conventional data management, we use a similar approach for data-intensive applications that incorporate humans. Specifically, declarative queries are posed over stored relational data as well as data computed on-demand from the crowd, and the underlying system orchestrates the computation of query answers.

We present Deco, a database system for declarative crowdsourcing. We describe Deco's data model, query language, and our prototype. Deco's data model was designed to be *general* (it can be instantiated to other proposed models), *flexible* (it allows methods for data cleansing and external access to be plugged in), and *principled* (it has a precisely-defined semantics). Syntactically, Deco's query language is a simple extension to SQL. Based on Deco's data model, we define a precise semantics for arbitrary queries involving both stored data and data obtained from the crowd. We then describe the Deco query processor which uses a novel push-pull hybrid execution model to respect the Deco semantics while coping with the unique combination of latency, monetary cost, and uncertainty introduced in the crowdsourcing environment. Finally, we experimentally explore the query processing alternatives provided by Deco using our current prototype.

Categories and Subject Descriptors

H.2.1 [Database Management]: Logical Design—*data models*;
H.2.3 [Database Management]: Languages—*query languages*

Keywords

declarative crowdsourcing, human computation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CIKM '12, October 29–November 2, 2012, Maui, HI, USA.
Copyright 2012 ACM 978-1-4503-1156-4/12/10...\$15.00.

```
CREATE TABLE Department (
  university STRING,
  name STRING,
  url CROWD STRING,
  phone STRING,
  PRIMARY KEY (university, name) );
```

such queries requires human input for providing information that is missing from the database, for performing computationally difficult functions, and for matching, ranking, or aggregating results based on fuzzy criteria. CrowdDB uses human input via crowdsourcing to process queries that neither database systems nor search engines can adequately answer. It uses SQL both as a language for posing complex queries and as a way to model data. While CrowdDB leverages many aspects of traditional database systems, there are also important differences. Conceptually, a major change is that the traditional closed-world assumption for query processing does not hold for human input. From an implementation perspective, human-oriented query operators are needed to solicit, integrate and cleanse crowdsourced data. Furthermore, performance and cost depend on a number of new factors including worker affinity, training, fatigue, motivation and location. We describe the design of CrowdDB, report on an initial set of experiments using Amazon Mechanical Turk, and outline important avenues for future work in the development of crowdsourced query processing systems.

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems

General Terms

Human Factors, Languages, Design, Performance

1. INTRODUCTION

Relational database systems have achieved widespread adoption, not only in the business environments for which they were originally envisioned, but also for many other types of structured data, such as personal, social, and even scientific information. Still, as data creation and use become increasingly democratized through web, mobile and other technologies, the limitations of the technology are becoming more apparent. RDBMSs make several key

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD '11, June 12–16, 2011, Athens, Greece.
 Copyright 2011 ACM 978-1-4503-0661-4/11/06...\$10.00.

Crowdsourced Databases: Query Processing with People

Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, Robert C. Miller
 MIT CSAIL
 {marcu, srrice, karger, madden, rcm}@csail.mit.edu

Amazon's Mechanical Turk service ("MTurk") allows users to outsource various kinds of HITs for filtering, aggregating, sorting, and joining data sources together is common, and comes with a set of challenges in optimizing the cost per HIT, the overall time to task completion, and the accuracy of MTurk results. We propose Quirk, a novel query system for managing these workflows, allowing crowd-powered processing of relational databases. We describe a number of query execution and optimization challenges, and discuss some potential solutions.

1. INTRODUCTION

Amazon's Mechanical Turk service (<https://www.mturk.com/mturk/welcome>) ("MTurk") allows users to post short tasks ("HITs") that other users ("turkers") can receive a small amount of money for completing. A HIT creator specifies how much he or she will pay for a completed task. Example HITs involve compiling some information from the web, labeling the subject of an image, or comparing two documents. More complicated tasks, such as ranking a set of ten items or completing a survey are also possible. MTurk is used widely to perform data analysis tasks which are either easier to express to humans than to computers, or for which there aren't yet effective artificial intelligence algorithms. Task prices vary from a few cents (\$.01-\$0.02/HIT is a common price) to several dollars for completing a survey. MTurk has around 100,000-300,000 HITs posted at any time (<http://mturk-tracker.com/general/>). Novel uses of MTurk include matching earthquake survivor pictures with missing persons in Haiti (<http://app.boextra.org/mission/show/missionid/605/mode/do>), authoring a picture book (<http://bjoern.org/projects/catbook/>), and embedding turkers as editors into a word processing system [2]. From the point of view of a database system, crowd-powered tasks can be seen as operators—similar to user-defined functions—that are invoked as a part of query processing.

This article is published under a Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0/>), which permits distribution and reproduction in any medium as well allowing derivative works, provided that you attribute the original work to the author(s) and CIDR 2011. CIDR 2011

For example, given a database storing a table of images, a user might want to query for images of flowers, generating a HIT per image to have turkers perform the filter task. Several challenges arise in processing such a workflow. First, each HIT can take minutes to return, requiring an asynchronous query executor. Second, in addition to considering time, a crowdworker-aware optimizer must consider monetary cost and result accuracy. Finally, the selectivity of operators can not be predicted *a priori*, requiring an adaptive approach to query processing.

In this paper, we propose Quirk, a crowdworker-aware database system which addresses these challenges. Quirk can issue HITs that extract, order, filter, and join complex datatypes, such as images and large text blobs. While we describe Quirk here using the language of MTurk (turkers and HITs), and our initial prototype runs on MTurk, we aim for Quirk to be crowd-platform-independent. Future versions of Quirk will compile tasks for different kinds of crowds with different interfaces and incentive systems. Quirk is a new system in active development; we describe our vision for the system, propose a high level sketch of a UDF-like syntax for executing these HITs, and explore a number of questions that arise in such a system, including:

- What is the HIT equivalent of various relational operations? For example, an equijoin HIT might require humans to identify equal items, whereas a HIT-based sort can use human comparators.
- How many HITs should be generated for a given task? For example, when sorting, one can generate a HIT that asks users to score many items, or to simply compare two. How can the system optimize each HIT?
- Given that a large database could result in millions of HITs, how should the system choose which HITs to issue? Given that only a fraction of items in a large database can have HITs generated for them, what is the proper notion of query correctness?
- How much to charge for a HIT? Higher prices can lead to faster answers. Can the system adaptively optimize the price of HITs based on user-specified response time and budget constraints?
- Who is the right crowdworker for a task? MTurk provides paid workers, but an enterprise might prefer expert workers, and an operation with a tight budget might look for non-monetary incentives.

2. MOTIVATING EXAMPLES

Here is a list of tasks that Quirk should be able to run:



Deco: Declarative Crowdsourcing

Aditya Parameswaran
 Stanford University
 Stanford, CA
 adityagg@cs.stanford.edu

Hyunjung Park
 Stanford University
 Stanford, CA
 hyunjung@cs.stanford.edu

Hector Garcia-Molina
 Stanford University
 Stanford, CA
 hector@cs.stanford.edu

Neoklis Polyzotis
 UC Santa Cruz
 Santa Cruz, CA
 alkis@cs.ucsc.edu

Jennifer Widom
 Stanford University
 Stanford, CA
 widom@cs.stanford.edu

ABSTRACT

Crowdsourcing enables programmers to incorporate "human computation" as a building block in algorithms that cannot be fully automated, such as text analysis and image recognition. Similarly, humans can be used as a building block in data-intensive applications—providing, comparing, and verifying data used by applications. Building upon the decades-long success of declarative approaches to conventional data management, we use a similar approach for data-intensive applications that incorporate humans. Specifically, declarative queries are posed over stored relational data as well as data computed on-demand from the crowd, and the underlying system orchestrates the computation of query answers.

We present Deco, a database system for declarative crowdsourcing. We describe Deco's data model, query language, and our prototype. Deco's data model was designed to be *general* (it can be instantiated to other proposed models), *flexible* (it allows methods for data cleansing and external access to be plugged in), and *principled* (it has a precisely-defined semantics). Syntactically, Deco's query language is a simple extension to SQL. Based on Deco's data model, we define a precise semantics for arbitrary queries involving both stored data and data obtained from the crowd. We then describe the Deco query processor which uses a novel push-pull hybrid execution model to respect the Deco semantics while coping with the unique combination of latency, monetary cost, and uncertainty introduced in the crowdsourcing environment. Finally, we experimentally explore the query processing alternatives provided by Deco using our current prototype.

Categories and Subject Descriptors

H.2.1 [Database Management]: Logical Design—*data models*;
 H.2.3 [Database Management]: Languages—*query languages*

Keywords

declarative crowdsourcing, human computation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CIKM '12, October 29–November 2, 2012, Maui, HI, USA.
 Copyright 2012 ACM 978-1-4503-1156-4/12/10...\$15.00.

```
CREATE TABLE Department (
  university STRING,
  name STRING,
  url CROWD STRING,
  phone STRING,
  PRIMARY KEY (university, name) );
```

Missing Data

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems

General Terms

Human Factors, Languages, Design, Performance

1. INTRODUCTION

Relational database systems have achieved widespread adoption, not only in the business environments for which they were originally envisioned, but also for many other types of structured data, such as personal, social, and even scientific information. Still, as data creation and use become increasingly democratized through web, mobile and other technologies, the limitations of the technology are becoming more apparent. RDBMSs make several key

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD '11, June 12–16, 2011, Athens, Greece.
 Copyright 2011 ACM 978-1-4503-0661-4/11/06...\$10.00.

Crowdsourced Databases: Query Processing with People

Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, Robert C. Miller
 MIT CSAIL
 {marcu, srrice, karger, madden, rcm}@csail.mit.edu

Amazon's Mechanical Turk service ("MTurk") allows users to post short tasks ("HITs") that other users ("turkers") can receive a small amount of money for completing. A HIT creator specifies how much he or she will pay for a completed task. Example HITs involve compiling some information from the web, labeling the subject of an image, or comparing two documents. More complicated tasks, such as ranking a set of ten items or completing a survey are also possible. MTurk is used widely to perform data analysis tasks which are either easier to express to humans than to computers, or for which there aren't yet effective artificial intelligence algorithms. Task prices vary from a few cents (\$.01-\$0.02/HIT is a common price) to several dollars for completing a survey. MTurk has around 100,000-300,000 HITs posted at any time (<http://mturk-tracker.com/general/>). Novel uses of MTurk include matching earthquake survivor pictures with missing persons in Haiti (<http://app.boextra.org/mission/show/missionid/605/mode/do>), authoring a picture book (<http://bjoern.org/projects/catbook/>), and embedding turkers as editors into a word processing system [2].

1. INTRODUCTION

Amazon's Mechanical Turk service (<https://www.mturk.com/mturk/welcome>) ("MTurk") allows users to post short tasks ("HITs") that other users ("turkers") can receive a small amount of money for completing. A HIT creator specifies how much he or she will pay for a completed task. Example HITs involve compiling some information from the web, labeling the subject of an image, or comparing two documents. More complicated tasks, such as ranking a set of ten items or completing a survey are also possible. MTurk is used widely to perform data analysis tasks which are either easier to express to humans than to computers, or for which there aren't yet effective artificial intelligence algorithms. Task prices vary from a few cents (\$.01-\$0.02/HIT is a common price) to several dollars for completing a survey. MTurk has around 100,000-300,000 HITs posted at any time (<http://mturk-tracker.com/general/>). Novel uses of MTurk include matching earthquake survivor pictures with missing persons in Haiti (<http://app.boextra.org/mission/show/missionid/605/mode/do>), authoring a picture book (<http://bjoern.org/projects/catbook/>), and embedding turkers as editors into a word processing system [2].

This article is published under a Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0/>), which permits distribution and reproduction in any medium as well allowing derivative works, provided that you attribute the original work to the author(s) and CIDR 2011. CIDR 2011



Deco: Declarative Crowdsourcing

Aditya Parameswaran
 Stanford University
 Stanford, CA
 adityagg@cs.stanford.edu

Hyunjung Park
 Stanford University
 Stanford, CA
 hyunjung@cs.stanford.edu

Hector Garcia-Molina
 Stanford University
 Stanford, CA
 hector@cs.stanford.edu

Neoklis Polyzotis
 UC Santa Cruz
 Santa Cruz, CA
 alkis@cs.ucsc.edu

Jennifer Widom
 Stanford University
 Stanford, CA
 widom@cs.stanford.edu

ABSTRACT

Crowdsourcing enables programmers to incorporate "human computation" as a building block in algorithms that cannot be fully automated, such as text analysis and image recognition. Similarly, humans can be used as a building block in data-intensive applications—providing, comparing, and verifying data used by applications. Building upon the decades-long success of declarative approaches to conventional data management, we use a similar approach for data-intensive applications that incorporate humans. Specifically, declarative queries are posed over stored relational data as well as data computed on-demand from the crowd, and the underlying system orchestrates the computation of query answers.

We present Deco, a database system for declarative crowdsourcing. We describe Deco's data model, query language, and our prototype. Deco's data model was designed to be *general* (it can be instantiated to other proposed models), *flexible* (it allows methods for data cleansing and external access to be plugged in), and *principled* (it has a precisely-defined semantics). Syntactically, Deco's query language is a simple extension to SQL. Based on Deco's data model, we define a precise semantics for arbitrary queries involving both stored data and data obtained from the crowd. We then describe the Deco query processor which uses a novel push-pull hybrid execution model to respect the Deco semantics while coping with the unique combination of latency, monetary cost, and uncertainty introduced in the crowdsourcing environment. Finally, we experimentally explore the query processing alternatives provided by Deco using our current prototype.

Categories and Subject Descriptors

H.2.1 [Database Management]: Logical Design—*data models*;
 H.2.3 [Database Management]: Languages—*query languages*

Keywords

declarative crowdsourcing, human computation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CIKM '12, October 29–November 2, 2012, Maui, HI, USA.
 Copyright 2012 ACM 978-1-4503-1156-4/12/10...\$15.00.

Comparison Using Commonsense Knowledge

```
CREATE TABLE Department (  
  university STRING,  
  name STRING,  
  url CROWD STRING,  
  phone STRING,  
  PRIMARY KEY (university, name) );
```

Crowdsourced Databases: Query Processing with Commonsense Knowledge

Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, Robert S. Sutton
MIT CSAIL
(marcua, sirtice, karger, madden, rcm)@csail.mit.edu

```
SELECT profile FROM department  
WHERE name ~="CS";
```

Stanford, CA
adityagg@cs.stanford.edu
Stanford, CA
hyunjung@cs.stanford.edu
Stanford, CA
hector@cs.stanford.edu

Neoklis Polyzotis
UC Santa Cruz
Santa Cruz, CA
alkis@cs.ucsc.edu

Jennifer Widom
Stanford University
Stanford, CA
widom@cs.stanford.edu

ABSTRACT

Crowdsourcing enables programmers to incorporate “human computation” as a building block in algorithms that cannot be fully automated, such as text analysis and image recognition. Similarly, humans can be used as a building block in data-intensive applications—providing, comparing, and verifying data used by applications. Building upon the decades-long success of declarative approaches to conventional data management, we use a similar approach for data-intensive applications that incorporate humans. Specifically, declarative queries are posed over stored relational data as well as data computed on-demand from the crowd, and the underlying system orchestrates the computation of query answers.

We present *Deco*, a database system for declarative crowdsourcing. We describe *Deco*'s data model, query language, and our prototype. *Deco*'s data model was designed to be *general* (it can be instantiated to other proposed models), *flexible* (it allows methods for data cleansing and external access to be plugged in), and *principled* (it has a precisely-defined semantics). Syntactically, *Deco*'s query language is a simple extension to SQL. Based on *Deco*'s data model, we define a precise semantics for arbitrary queries involving both stored data and data obtained from the crowd. We then describe the *Deco* query processor which uses a novel push-pull hybrid execution model to respect the *Deco* semantics while coping with the unique combination of latency, monetary cost, and uncertainty introduced in the crowdsourcing environment. Finally, we experimentally explore the query processing alternatives provided by *Deco* using our current prototype.

Categories and Subject Descriptors

H.2.1 [Database Management]: Logical Design—*data models*;
H.2.3 [Database Management]: Languages—*query languages*

Keywords

declarative crowdsourcing, human computation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD '11, June 12–16, 2011, Athens, Greece.
Copyright 2011 ACM 978-1-4503-0661-4/11/06...\$10.00.

such queries requires human input for providing information that is missing from the database, for performing computationally difficult functions, and for matching, ranking, or aggregating results based on fuzzy criteria. CrowdDB uses human input via crowdsourcing to answer queries that neither database systems nor search engines can answer. It uses SQL both as a language for posing complex queries and as a way to model data. While CrowdDB leverages many aspects of traditional database systems, there are also several differences. Concretely, a major change is that CrowdDB uses SQL to pose queries that are not expressible in SQL. In particular, it uses SQL to integrate data from multiple sources, and to depend on a number of new factors including user affinity, training, fatigue, motivation and location. We describe the design of CrowdDB, report on an initial set of experiments using Amazon Mechanical Turk, and outline important avenues for future work in the development of crowdsourced query processing systems.

Missing Data

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems

General Terms

Human Factors, Languages, Design, Performance

1. INTRODUCTION

Relational database systems have achieved widespread adoption, not only in the business environments for which they were originally envisioned, but also for many other types of structured data, such as personal, social, and even scientific information. Still, as data creation and use become increasingly democratized through web, mobile and other technologies, the limitations of the technology are becoming more apparent. RDBMSs make several key

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD '11, June 12–16, 2011, Athens, Greece.
Copyright 2011 ACM 978-1-4503-0661-4/11/06...\$10.00.

questions, if they return any answers at all.

1.1 Power to the People

One obvious situation where existing systems produce wrong answers is when they are missing information required for answering the question. For example, the query:

```
SELECT market_capitalization FROM company  
WHERE name ~="I.B.M.";
```

returns an empty result if the company table instance in the database does not contain a record for “I.B.M.”. Of course, in reality, there are many reasons why such a record may be missing. For example, a data entry mistake may have omitted the I.B.M. record or the record may have been inadvertently deleted. Another possibility is that the record was entered incorrectly, say, as “L.B.N.”.

Traditional systems can erroneously return an empty result even when no errors are made. For example, if the record was entered correctly, but the name used was “International Business Machines” rather than “I.B.M.” This latter “entity resolution” problem is not due to an error but is simply an artifact of having multiple ways to refer to the same real-world entity.

There are two fundamental problems at work here. First, relational database systems are based on the “Closed World Assumption”: information that is not in the database is considered to be false or non-existent. Second, relational databases are extremely literal. They expect that data has been properly cleaned and validated before entry and do not natively tolerate inconsistencies in data or queries.

As another example, consider a query to find the best among a collection of images to use in a motivational slide show:

```
SELECT image FROM picture  
WHERE topic = "Business Success"  
ORDER BY relevance LIMIT 1;
```

In this case, unless the relevance of pictures to specific topics has been previously obtained and stored, there is simply no good way to ask this question of a standard RDBMS. The issue here is one of judgement; one cannot reliably answer the question simply by applying relational operators on the database.

All of the above queries, however, while unanswerable by today's relational database systems, could easily be answered by people, especially people who have access to the Internet.

Amazon's Mechanical Turk service (https://www.mturk.com/mturk/welcome) (“MTurk”) allows users to post short tasks (“HITS”) that other users (“turkers”) can receive a small amount of money for completing. A HIT creator specifies how much he or she will pay for a completed task. Example HITS involve compiling some information from the web, labeling the subject of an image, or comparing two documents. More complicated tasks, such as ranking a set of ten items or completing a survey are also possible. MTurk is used widely to perform data analysis tasks which are either easier to express to humans than to computers, or for which there aren't yet effective artificial intelligence algorithms.

Task prices vary from a few cents (\$.01-\$0.02/HIT is a common price) to several dollars for completing a survey. MTurk has around 100,000-300,000 HITS posted at any time (http://mturk-tracker.com/general/). Novel uses of MTurk include matching earthquake survivor pictures with missing persons in Haiti (http://app.beextra.org/mission/show/missionid/605/mode/do), authoring a picture book (http://bjoern.org/projects/catbook/), and embedding turkers as editors into a word processing system [2].

1. INTRODUCTION

Amazon's Mechanical Turk service (https://www.mturk.com/mturk/welcome) (“MTurk”) allows users to post short tasks (“HITS”) that other users (“turkers”) can receive a small amount of money for completing. A HIT creator specifies how much he or she will pay for a completed task. Example HITS involve compiling some information from the web, labeling the subject of an image, or comparing two documents. More complicated tasks, such as ranking a set of ten items or completing a survey are also possible. MTurk is used widely to perform data analysis tasks which are either easier to express to humans than to computers, or for which there aren't yet effective artificial intelligence algorithms.

Task prices vary from a few cents (\$.01-\$0.02/HIT is a common price) to several dollars for completing a survey. MTurk has around 100,000-300,000 HITS posted at any time (http://mturk-tracker.com/general/). Novel uses of MTurk include matching earthquake survivor pictures with missing persons in Haiti (http://app.beextra.org/mission/show/missionid/605/mode/do), authoring a picture book (http://bjoern.org/projects/catbook/), and embedding turkers as editors into a word processing system [2].

From the point of view of a database system, crowd-powered tasks can be seen as operators—similar to user-defined functions—that are invoked as a part of query processing. For example, given a database storing a table of images, a user might want to query for images of flowers, generating a HIT per image to have turkers perform the filter task. Several challenges arise in processing such a workflow. First, each HIT can take minutes to return, requiring an asynchronous query executor. Second, in addition to considering time, a crowdworker-aware optimizer must consider monetary cost and result accuracy. Finally, the selectivity of operators can not be predicted *a priori*, requiring an adaptive approach to query processing.

This article is published under a Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/), which permits distribution and reproduction in any medium as well allowing derivative works, provided that you attribute the original work to the author(s) and CIDR 2011. CIDR 2011

cessing. For example, given a database storing a table of images, a user might want to query for images of flowers, generating a HIT per image to have turkers perform the filter task. Several challenges arise in processing such a workflow. First, each HIT can take minutes to return, requiring an asynchronous query executor. Second, in addition to considering time, a crowdworker-aware optimizer must consider monetary cost and result accuracy. Finally, the selectivity of operators can not be predicted *a priori*, requiring an adaptive approach to query processing.

In this paper, we propose *Qurk*, a crowdworker-aware database system which addresses these challenges. *Qurk* can issue HITS that extract, order, filter, and join complex datatypes, such as images and large text blobs. While we describe *Qurk* here using the language of MTurk (turkers and HITS), and our initial prototype runs on MTurk, we aim for *Qurk* to be crowd-platform-independent. Future versions of *Qurk* will compile tasks for different kinds of crowds with different interfaces and incentive systems. *Qurk* is a new system in active development; we describe our vision for the system, propose a high level sketch of a UDF-like syntax for executing these HITS, and explore a number of questions that arise in such a system, including:

- What is the HIT equivalent of various relational operations? For example, an equijoin HIT might require humans to identify equal items, whereas a HIT-based sort can use human comparators.
- How many HITS should be generated for a given task? For example, when sorting, one can generate a HIT that asks users to score many items, or to simply compare two. How can the system optimize each HIT?
- Given that a large database could result in millions of HITS, how should the system choose which HITS to issue? Given that only a fraction of items in a large database can have HITS generated for them, what is the proper notion of query correctness?
- How much to charge for a HIT? Higher prices can lead to faster answers. Can the system adaptively optimize the price of HITS based on user-specified response time and budget constraints?
- Who is the right crowdworker for a task? MTurk provides paid workers, but an enterprise might prefer expert workers, and an operation with a tight budget might look for non-monetary incentives.

2. MOTIVATING EXAMPLES

Here is a list of tasks that *Qurk* should be able to run:

Comparison Using Commonsense Knowledge

```
CREATE TABLE Department (  
  university STRING,  
  name STRING,  
  url CROWD STRING,  
  phone STRING,  
  PRIMARY KEY (university, name) );
```

Crowdsourced Databases: Query Processing with Commonsense Knowledge

Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, Robert H. Korth
MIT CSAIL
(marcua, sirrice, karger, madden, rcm)@csail.mit.edu

```
SELECT profile FROM department  
WHERE name ~="CS";
```

Missing Data

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems

General Terms

Human Factors, Languages, Design, Performance

1. INTRODUCTION

Relational database systems have achieved widespread adoption, not only in the business environments for which they were originally envisioned, but also for many other types of structured data, such as personal, social, and even scientific information. Still, as data creation and use become increasingly democratized through web, mobile and other technologies, the limitations of the technology are becoming more apparent. RDBMSs make several key

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD '11, June 12–16, 2011, Athens, Greece.
Copyright 2011 ACM 978-1-4503-0661-4/11/06...\$10.00.

61

1.1 Power to the People

One obvious situation where existing systems produce wrong answers is when they are missing information required for answering the question. For example, the query:

```
SELECT market_capitalization FROM company  
WHERE name ~="IBM";
```

returns an empty result even when the company table instance in the database does contain a record for "IBM". Of course, in reality, there are many reasons why such a record may be missing. For example, a data entry mistake may have omitted the IBM record or the record may have been inadvertently deleted. Another possibility is that the record was entered incorrectly, say, as "I.B.N."

Traditional systems can erroneously return an empty result even when no errors are made. For example, if the record was entered correctly, but the name used was "International Business Machines" rather than "IBM". This latter "entity resolution" problem is not due to an error but is simply an artifact of having multiple ways to refer to the same real-world entity.

There are two fundamental problems at work here. First, relational database systems are based on the "Closed World Assumption": information that is not in the database is considered to be false or non-existent. Second, relational databases are extremely literal. They expect that data has been properly cleaned and validated before entry and do not natively tolerate inconsistencies in data or queries.

As another example, consider a query to find the best among a collection of images to use in a motivational slide show:

```
SELECT image FROM picture  
WHERE topic = "Business Success"  
ORDER BY relevance LIMIT 1;
```

In this case, unless the relevance of pictures to specific topics has been previously obtained and stored, there is simply no good way to ask this question of a standard RDBMS. The issue here is one of judgement; one cannot reliably answer the question simply by applying relational operators on the database.

All of the above queries, however, while unanswerable by today's relational database systems, could easily be answered by people, especially people who have access to the Internet.

Amazon's Mechanical Turk service (https://www.mturk.com/mturk/welcome) ("MTurk") allows users to post short tasks ("HITs") that other users ("turkers") can receive a small amount of money for completing. A HIT creator specifies how much he or she will pay for a completed task. Example HITs involve compiling some information from the web, labeling the subject of an image, or comparing two documents. More complicated tasks, such as ranking a set of ten items or completing a survey are also possible. MTurk is used widely to perform data analysis tasks which are either easier to express to humans than to computers, or for which there aren't yet effective artificial intelligence algorithms.

Task prices vary from a few cents (\$.01-\$0.02/HIT is a common price) to several dollars for completing a survey. MTurk has around 100,000-300,000 HITs posted at any time (http://mturk-tracker.com/general/). Novel uses of MTurk include matching earthquake survivor pictures with missing persons in Haiti (http://app.beextra.org/mission/show/missionid/605/mode/do), authoring a picture book (http://bjoern.org/projects/catbook/), and embedding turkers as editors into a word processing system (http://www.mturk.com/mturk/welcome).

1. INTRODUCTION

Amazon's Mechanical Turk service (https://www.mturk.com/mturk/welcome) ("MTurk") allows users to post short tasks ("HITs") that other users ("turkers") can receive a small amount of money for completing. A HIT creator specifies how much he or she will pay for a completed task. Example HITs involve compiling some information from the web, labeling the subject of an image, or comparing two documents. More complicated tasks, such as ranking a set of ten items or completing a survey are also possible. MTurk is used widely to perform data analysis tasks which are either easier to express to humans than to computers, or for which there aren't yet effective artificial intelligence algorithms.

Task prices vary from a few cents (\$.01-\$0.02/HIT is a common price) to several dollars for completing a survey. MTurk has around 100,000-300,000 HITs posted at any time (http://mturk-tracker.com/general/). Novel uses of MTurk include matching earthquake survivor pictures with missing persons in Haiti (http://app.beextra.org/mission/show/missionid/605/mode/do), authoring a picture book (http://bjoern.org/projects/catbook/), and embedding turkers as editors into a word processing system (http://www.mturk.com/mturk/welcome).

```
SELECT p FROM picture  
WHERE subject = "Golden Gate Bridge"
```

211

Multimodal Data

For example, given a database storing a table of images, a user might want to query for images of flowers, generating a HIT per image to have turkers perform the filter task. Several challenges arise in processing such a workflow. First, each HIT can take minutes to return, requiring an asynchronous query executor. Second, in addition to considering time, a crowdworker-aware optimizer must consider monetary cost and result accuracy. Finally, the selectivity of operators can not be predicted *a priori*, requiring an adaptive approach to query processing.

In this paper, we propose Quirk, a crowdworker-aware database system which addresses these challenges. Quirk can issue HITs that extract, order, filter, and join complex datatypes, such as images and large text blobs. While we describe Quirk here using the language of MTurk (turkers and HITs), and our initial prototype runs on MTurk, we aim for Quirk to be crowd-platform-independent. Future versions of Quirk will compile tasks for different kinds of crowds with different interfaces and incentive systems. Quirk is a new system in active development; we describe our vision for the system, propose a high level sketch of a UDF-like syntax for executing these HITs, and explore a number of questions that arise in such a system, including:

- What is the HIT equivalent of various relational operations? For example, an equijoin HIT might require humans to identify equal items, whereas a HIT-based sort can use human comparators.
- How many HITs should be generated for a given task? For example, when sorting, one can generate a HIT that asks users to score many items, or to simply compare two. How can the system optimize each HIT?
- Given that a large database could result in millions of HITs, how should the system choose which HITs to issue? Given that only a fraction of items in a large database can have HITs generated for them, what is the proper notion of query correctness?
- How much to charge for a HIT? Higher prices can lead to faster answers. Can the system adaptively optimize the price of HITs based on user-specified response time budget constraints?

ABSTRACT

Crowdsourcing enables programmers to incorporate "human computation" as a building block in algorithms that cannot be fully automated, such as text analysis and image recognition. Similarly, humans can be used as a building block in data-intensive applications—providing, comparing, and verifying data used by applications. Building upon the decades-long success of declarative approaches to conventional data management, we use a similar approach for data-intensive applications that incorporate humans. Specifically, declarative queries are posed over stored relational data as well as data computed on-demand from the crowd, and the underlying system orchestrates the computation of query answers.

We present Deco, a database system for declarative crowdsourcing. We describe Deco's data model, query language, and our prototype. Deco's data model was designed to be *general* (it can be instantiated to other proposed models), *flexible* (it allows methods for data cleansing and external access to be plugged in), and *principled* (it has a precisely-defined semantics). Syntactically, Deco's query language is a simple extension to SQL. Based on Deco's data model, we define a precise semantics for arbitrary queries involving both stored data and data obtained from the crowd. We then describe the Deco query processor which uses a novel push-pull hybrid execution model to respect the Deco semantics while coping with the unique combination of latency, monetary cost, and uncertainty introduced in the crowdsourcing environment. Finally, we experimentally explore the query processing alternatives provided by Deco using our current prototype.

Categories and Subject Descriptors

H.2.1 [Database Management]: Logical Design—*data models*;
H.2.3 [Database Management]: Languages—*query languages*

Keywords

declarative crowdsourcing, human computation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD '11, June 12–16, 2011, Athens, Greece.
Copyright 2011 ACM 978-1-4503-0661-4/11/06...\$10.00.

1203

CAESURA: Language Models as Multi-Modal Query Planners

Matthias Urban
Technical University of Darmstadt

Carsten Binnig
Technical University of Darmstadt & DFKI

ABSTRACT

Traditional query planners translate SQL queries into query plans to be executed over relational data. However, it is impossible to query other data modalities, such as images, text, or video stored in modern data systems such as data lakes using these query planners. In this paper, we propose Language-Model-Driven Query Planning, a new paradigm of query planning that uses Language Models to translate natural language queries into executable query plans. Different from relational query planners, the resulting query plans can contain complex operators that are able to process arbitrary modalities. As part of this paper, we present a first GPT-4 based prototype called CAESURA and show the general feasibility of this idea on two datasets. Finally, we discuss several ideas to improve the query planning capabilities of today’s Language Models.

1 INTRODUCTION

Query planning, the basic process of deriving an executable query plan in response to a user query, has conceptually stayed essentially the same since IBM’s System R was introduced in 1974 [14]. In traditional DBMSs, a logical plan is first obtained from parsing a SQL query and then optimized by improving the order of the query’s operators. In a second step, each logical operator is mapped to a concrete implementation to obtain a physical plan, which is eventually executed. In the past decades, research has focused primarily on improving the efficiency of query plans by focusing various individual aspects (e.g. the cost model [3]).

However, this traditional approach to query planning is fundamentally limited to two important aspects: Firstly, it only applies to query languages such as SQL, where semantics of queries are clear and can be easily parsed into (at least canonical) query plan to execute the query. Secondly, it is only possible to query structured relational data stored in tables. Due to these limitations and several new trends, we argue that it is finally time to re-think how query planning is done.

Trend 1: Multi-Modal Data. In today’s industries, huge amounts of non-relational multi-modal data (e.g. images, documents, sensor data, ...) need to be stored and processed, which has led to solutions that store data outside classical databases. For instance, medical clinics need to store and analyze MRI scans and patient reports along with structured patient metadata. Since these data modalities cannot be stored and queried easily in today’s databases, they are usually stored in data lakes, which, however, makes gaining insights from these modalities hard. To gain insights from such multi-modal data lakes, the data needs to be made accessible first, usually by manually constructing complex data processing pipelines.

This paper is published under the Creative Commons Attribution 4.0 International License. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution, provided that you attribute the original work to the author(s) and CDBR 2025, 19th Annual Conference on Innovative Data Systems Research (CIDR ’25), January 19–21, 2024, Chalmers, USA.

While recently several pioneering systems have been proposed to ease the querying of multi-modal data in data lakes [3, 7, 16, 18, 19], these systems come with significant limitations. For instance, the queries supported by these systems are often limited in complexity (e.g. only simple queries with a single value as answer are supported [3]), or non-relational modalities are only used as filters [7], or they are limited to only a very few modalities [16, 19]. An ideal data system for multi-modal data, instead, would allow all types of queries and automatically construct the data processing pipelines, that have previously been constructed manually. This would allow users to formulate queries that combine information across modalities, and let them gain insights in a few minutes, for which they previously would have needed several days or weeks.

Trend 2: Natural Language Interfaces. SQL, with its declarative nature, has initially been designed to be understandable by laypersons. In practice, however, formulating complex queries in SQL requires profound knowledge of the language, making databases not accessible to domain experts and management staff. Usually, these persons are required to interact with specifically trained data scientists to obtain insights from data stored in databases, a process that often requires many iterations.

Hence, in recent years, Natural Language Interfaces for databases have emerged, which would allow laypersons to query databases using natural language. However, existing approaches typically translate natural language into SQL [28, 29] and are therefore limited to relational data. Another direction are question-answering systems which work on modalities beyond tables [3]. However, question-answering systems only support queries that are much less expressive than what can be done with SQL.

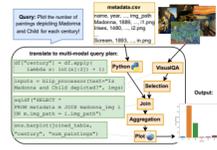


Figure 1: Example illustrating how a natural language query is automatically translated into a multi-modal query plan containing relational operators, machine generated Python UDFs, and a VisualQA Machine Learning model. The output is presented as a plot, making it easy and fast to gain insights from multi-modal data. The green boxes show code snippets that are executed when executing the plan.

Palimpzest: Optimizing AI-Powered Analytics with Declarative Query Processing

Chunwei Liu¹, Matthew Russo², Michael Cafarella,
Lei Cao³, Peter Baile Chen, Zui Chen, Michael Franklin¹,
Tim Kraska, Samuel Madden, Rami Shihoub, Gerardo Vitagliano
¹MIT, ²University of Arizona, ³University of Chicago, ⁴Harvard University
chunwei,mdrusso,michjc,madden,gerarvit@csail.mit.edu, peterse,chen423,kraska@mit.edu,
cao lei@arizona.edu, mjfranklin@uchicago.edu, rana@seas.harvard.edu

ABSTRACT

A long-standing goal of data management systems has been to build systems which can compute quantitative insights over large collections of unstructured data in a cost-effective manner. Until recently, it was difficult and expensive to extract facts from company documents, data from scientific papers, or metrics from image and video corpora. Today’s models can accomplish these tasks with high accuracy. However, a programmer who wants to answer a substantive AI-powered query must orchestrate large numbers of models, prompts, and data operations. In this paper, we present PALIMPZEST, a system that enables programmers to pose AI-powered analytical queries over arbitrary collections of unstructured data in a simple declarative language. The system uses a cost optimization framework—which explores the search space of AI models, prompting techniques, and related foundation model optimizations. PALIMPZEST implements the query while navigating the trade-offs between runtime, financial cost, and output data quality. We introduce a novel language for AI-powered analytics tasks, the optimization methods that PALIMPZEST uses, and the prototype system itself. We evaluate PALIMPZEST on a real-world workload. Our system produces plans that are up to 3.3x faster and 2.9x cheaper than a baseline method when using a single-literal setup, while also achieving superior F1 scores. PALIMPZEST applies its optimizations automatically, requiring no additional work from the user.

1 INTRODUCTION

Advances in AI models have driven progress in applications such as question answering [11, 25], chatbots [5], autonomous agents [17, 18], and code synthesis [7, 9]. In many cases, these systems have evolved far beyond posing a simple question to a chat model; they are compound AI systems [24] that combine elements of data processing, such as Retrieval-Augmented Generation (RAG); ensembles of different models; multi-step chain-of-thought reasoning; and in many cases, cloud-based models. It is easy for the runtime, cost, and complexity of these AI systems to escalate quickly, particularly when applied to large collections of documents.

The performance gap between traditional data processing components and AI-powered components is profound. Naïvely scaling AI systems to process workloads with thousands or millions of inputs

This paper is published under the Creative Commons Attribution 4.0 International License. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution, provided that you attribute the original work to the author(s) and CDBR 2025, 19th Annual Conference on Innovative Data Systems Research (CIDR ’25), January 19–21, 2024, Chalmers, The Netherlands.

requires spending a large amount of runtime and money executing high-end AI models. For instance, a high-quality open source LLM on a modern GPU processes about 100-125 tokens per second, yielding a throughput of less than 1 KB per second, assuming that each token averages 7 bytes. OpenAI’s GPT-4o mini model costs 15 cents for 1M input tokens, equivalent to processing 5MB of data. These numbers are many orders of magnitude worse than any other component of the modern data processing stack, such as storage, network bandwidth, SQL query processing time, and so on. Thus, optimizing the use of AI components is crucial. Meanwhile, current AI infrastructure is in tremendous technical flux. New models and implementation techniques are published weekly, while model costs and runtimes change frequently. Harnessing the latest advances in model runtime, cost, and quality is complex, error-prone, and requires developers of AI applications to constantly rewrite and retune their systems.

AI engineers face a variety of technical decisions, including optimizing prompt wording and strategy (e.g. chain-of-thought, ReAct [23]), selecting the best model for each subtask while balancing time, cost and quality, and deciding on the best implementation approach for each subtask (e.g. using a foundation model query, synthesized code, locally trained model, or a mixture of agents [20]). Additionally, engineers must manage GPU cache and memory utilization, handle LLM context limits, design efficient execution plans for scaling to larger datasets, and integrate parallelized components for optimal system efficiency. Decisions on integration with external data systems also require careful parameter selection to achieve the best trade-offs between speed, cost, and quality.

The space of possible decisions is vast, and choosing wisely depends on low-level details of the exact task. Moreover, the definition of “best” can change over time: a developer might prefer “cheap and low-quality” execution when quickly testing initial proof-of-concept ideas, then switch to “costly but high-quality” for customer deployment. Finally, the changing technical landscape means that the optimization choices made today might be obsolete tomorrow.

Our Goal: The key insight is that machines, not human engineers, should decide how best to optimize AI applications. Engineers should be able to write AI programs at a high level of abstraction and rely on an optimizer to find a physical implementation that best fits their use case. This concept echoes the circumstances that led to the creation of the relational database query optimizer in the 1970s—a period marked by the need for performance enhancements amid significant technological shifts. While today’s technical challenges differ, the principle of declarative program optimization remains profoundly relevant.

ThalamusDB: Approximate Query Processing on Multi-Modal Data

SAEHAN JO, Cornell University, USA
IMMANUEL TRUMMER, Cornell University, USA

We introduce ThalamusDB, a novel approximate query processing system that processes complex SQL queries on multi-modal data. ThalamusDB supports SQL queries integrating natural language predicates on visual, audio, and text data. To answer such queries, ThalamusDB exploits a collection of zero-shot models in combination with relational processing. ThalamusDB utilizes deterministic approximate query processing, harnessing the relative efficiency of relational processing to mitigate the computational demands of machine learning inference. For evaluating a natural language predicate, ThalamusDB requests a small number of labels from users. User can specify their preferences on the performance objective regarding the three relevant metrics: approximation error, computation time, and labeling overheads. The ThalamusDB query optimizer chooses optimized plans according to user preferences, prioritizing data processing and requested labels to maximize impact. Experiments with several real-world data sets, taken from Craigslist, YouTube, and Netflix, show that ThalamusDB achieves an average speedup of 35.0x over MindsDB, an exact processing baseline, and outperforms ABAE, a sampling-based method, in 78.9% of cases.

CCS Concepts: • Information systems — Online analytical processing engines; Data model extensions; • Computing methodologies — Machine learning.

Additional Key Words and Phrases: query processing, multi-modal data, neural models

ACM Reference Format:

Saehan Jo and Immanuel Trummer. 2024. ThalamusDB: Approximate Query Processing on Multi-Modal Data. Proc. ACM Manag. Data 2, 3 (SIGMOD), Article 186 (June 2024), 26 pages. <https://doi.org/10.1145/3654989>

1 INTRODUCTION

Prior work has introduced systems that integrate machine learning with databases, such as MindsDB [30] and EvalDB [17]. Using large neural models, these systems enable users to handle multi-modal data within a unified framework. In this context, the primary challenge in processing queries on multi-modal data is the computational cost associated with model inference, as shown in Figure 1. Processing large amounts of multi-modal data is very expensive and becomes the dominant factor in query processing. This processing overhead underscores the need for a more efficient solution. Hence, we introduce ThalamusDB, an approximate query processing (AQP) system designed for complex queries on multi-modal data. The code of ThalamusDB is available at <https://github.com/saehanjo/thalamusdb>.

ThalamusDB supports retrieval and aggregation queries (with joins) that integrate natural language predicates on multi-modal data. It supports an extended, relational data model, introducing specialized column types to integrate picture and audio data. Queries are formulated using an SQL dialect that supports predicates, formulated in natural language on text, picture, or audio data—

Authors’ Contact Information: Saehan Jo, Cornell University, Ithaca, New York, USA, sj683@cornell.edu; Immanuel Trummer, Cornell University, Ithaca, New York, USA, itrummer@cornell.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 2836-6573/2024/6-ART186
<https://doi.org/10.1145/3654989>

Proc. ACM Manag. Data, Vol. 2, No. 3 (SIGMOD), Article 186. Publication date: June 2024.



Semantic Operators: A Declarative Model for Rich, AI-based Data Processing

Liana Patel¹, Siddharth Jha¹, Melissa Pan², Harshit Gupta¹, Parth Asawa¹,
Carlos Guestrin¹, Matei Zaharia¹
¹Stanford University, ²UC Berkeley
Correspondence: lianapat@stanford.edu
LOTUS Repository and Tutorials

ABSTRACT

The semantic capabilities of large language models (LLMs) have the potential to enable rich analytics and reasoning over vast knowledge corpora. Unfortunately, existing systems either empirically optimize expensive LLM-powered operations with no performance guarantees, or serve a limited set of row-wise LLM operations, providing limited robustness, expressiveness and usability. We introduce semantic operators, the first formalism for declarative and general-purpose AI-based transformations based on natural language specifications (e.g. filtering, sorting, joining or aggregating records using natural language criteria). Each operator opens a rich space for execution plans, similar to relational operators. Our model specifies the expected behavior of each operator with a high-quality gold algorithm, and we develop an optimization framework that reduces cost, while providing accuracy guarantees with respect to a gold algorithm. Using this approach, we propose several novel optimizations to accelerate semantic filtering, joining, group-by and top-k operations by up to 1,008x. We implement semantic operators in the LOTUS system and demonstrate LOTUS’ effectiveness on real, bulk-semantic processing applications, including fact-checking, biomedical multi-label classification, search, and topic analysis. We show that the semantic operator model is expressive, capturing state-of-the-art AI pipelines in a few operator calls, and making it easy to express new pipelines that match or exceed quality of recent LLM-based analytic systems by up to 176%, while offering accuracy guarantees. Overall, LOTUS programs match or exceed the accuracy of state-of-the-art AI pipelines for each task while running up to 3.6x faster than the highest-quality baselines. LOTUS is publicly available at <https://github.com/lotus-data/lotus>.

1 INTRODUCTION

The powerful semantic capabilities of modern language models (LLMs) create exciting opportunities for building AI-based analytics systems that reason over vast knowledge corpora. Many applications require complex reasoning over large amounts of data, including both unstructured and structured data. For example, a researcher reviewing recent ArXiv [1] preprints may want to quickly obtain a summary of relevant papers from the past week, or find the papers that report the best performance for a particular task and dataset. Similarly, a medical professional may automatically extract biomedical characteristics and candidate diagnoses from many patient reports [2]. Likewise, organizations wish to automatically digest lengthy transcripts from internal meeting transcripts and chat histories to validate hypotheses about their business [4].

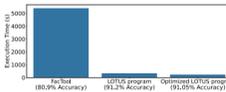


Figure 1: Execution time (y-axis) and accuracy, shown in parentheses, for 3 fact-checking implementations: (1) Fast Tool [21], a recent state-of-the-art research work, (ii) a short, un-optimized LOTUS program, and (iii) the same LOTUS program optimized with accuracy guarantees. Section 5 provides our full methodology.

Each of these tasks require a form of bulk semantic processing, where the analytics system must process large amounts of data and apply semantic-based analysis across a whole dataset. Supporting the full generality of these applications with efficient and easy-to-use analytics systems would have a transformative impact, similar to what RDBMSes had for tabular data. This prospect, however, raises two challenging questions: first, how should developers express semantic queries, and secondly, how should we design the underlying analysis system to achieve high efficiency and accuracy.

Unfortunately, existing systems are insufficient for bulk semantic processing, either limiting their expressiveness to row-wise LLM execution models or focusing on empirical optimizations without any formalism or accuracy guarantees. Overall, LOTUS programs match or exceed the accuracy of state-of-the-art AI pipelines for each task while running up to 3.6x faster than the highest-quality baselines. LOTUS is publicly available at <https://github.com/lotus-data/lotus>.

In this demo, we introduce SwellDB, a data system that follows a new architecture that enables query execution over dynamically generated tables. Unlike traditional systems which rely on previously stored data, SwellDB synthesizes tables on the fly based on input queries.

We address this question by introducing a data system architecture that enables query execution over dynamically generated tables, according to the input query. Our system, SwellDB, leverages Large Language Models to tackle the table generation problem as follows. Given a table definition consisting of a prompt (describing the table content) and a schema (a set of attributes and their descriptions), SwellDB instructs the LLM to compute a table generation plan. This plan specifies (1) which input data sources can be used to retrieve portions of the data, (2) how to utilize these

SwellDB: Dynamic Query-Driven Table Generation with Large Language Models

Victor Giannakouris
Cornell University,
Ithaca, NY, USA
vg292@cornell.edu

Immanuel Trummer
Cornell University,
Ithaca, NY, USA
itr224@cornell.edu

Abstract

We present SwellDB¹, a data system that enables analytical query processing over tables generated with Large Language Models (LLMs). Typically, traditional database systems function under the closed-world assumption, allowing query execution over already existing tables. In contrast, SwellDB dynamically generates the required tables based on input SQL queries, and user-defined table definitions. Powered by an LLM, SwellDB integrates external data sources, including diverse datasets, database systems, and search engines to extract and blend relevant information into a structured table. Given an input query and schema, SwellDB intelligently selects data sources, synthesizes structured tables adhering to the specified schema, and makes them queryable through its execution engine. During the demonstration, we will showcase how SwellDB addresses data generation and integration challenges across domains such as web search and bioinformatics. Attendees can define custom schemas and execute SQL queries, gaining hands-on experience with SwellDB’s table generation capabilities.

Consequently, end-users are often burdened with developing custom Extract-Transform-Load (ETL) workflows to create structured tables, enabling them to query this data using SQL. Large Language Models. LLMs offer a compelling solution to the challenges of information integration and table generation. Trained on vast amounts of data embedded within their weights, LLMs excel at data generation and augmentation tasks. For example, given input datasets and a search engine, an LLM can determine how to utilize those data sources to generate a table with specific content and schema. LLMs can efficiently answer questions such as which code or SQL query to execute to retrieve a portion of the requested data from the input data sources, or which search queries to use in order to retrieve data not available in the LLM or input sources. Additionally, LLMs can serve as preprocessors, transforming unstructured data (e.g., web-retrieved information) into structured formats that are queryable with SQL.

Taking these considerations into account, we pose the following question:

Given a set of input data sources, how can we leverage Large Language Models to dynamically generate tables of any content based on an input query?

We address this question by introducing a data system architecture that enables query execution over dynamically generated tables, according to the input query. Our system, SwellDB, leverages Large Language Models to tackle the table generation problem as follows. Given a table definition consisting of a prompt (describing the table content) and a schema (a set of attributes and their descriptions), SwellDB instructs the LLM to compute a table generation plan. This plan specifies (1) which input data sources can be used to retrieve portions of the data, (2) how to utilize these

Database, Large Language Models, Table Generation, ETL.

ACM Reference Format:
Victor Giannakouris and Immanuel Trummer. 2025. SwellDB: Dynamic Query-Driven Table Generation with Large Language Models. In *Companion of the 2025 International Conference on Management of Data (SIGMOD-Companion ’25)*, June 22–27, 2025, Berlin, Germany. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/372212.3725136>

1 Introduction

In this demo, we introduce SwellDB, a data system that follows a new architecture that enables query execution over dynamically generated tables. Unlike traditional systems which rely on previously stored data, SwellDB synthesizes tables on the fly based on input queries.

We address this question by introducing a data system architecture that enables query execution over dynamically generated tables, according to the input query. Our system, SwellDB, leverages Large Language Models to tackle the table generation problem as follows. Given a table definition consisting of a prompt (describing the table content) and a schema (a set of attributes and their descriptions), SwellDB instructs the LLM to compute a table generation plan. This plan specifies (1) which input data sources can be used to retrieve portions of the data, (2) how to utilize these

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third party components of this work must be honored. For all other uses, contact the owner/author(s).
SIGMOD-Companion ’25, June 22–27, 2025, Berlin, Germany
© 2025 Copyright held by the owner/author(s).
ACM ISBN 978-4-401-7161-8/2025/6
<https://doi.org/10.1145/372212.3725136>

CAESURA: Language Models as Multi-Modal Query Planners

Matthias Urban, Carsten Binnig

ABSTRACT
Traditional query planners translate SQL queries into query plans to be executed over relational data. However, it is impossible to query other data modalities, such as images, text, or video stored in modern data systems such as data lakes using these query planners. In this paper, we propose Language-Model-Driven Query Planning, a new paradigm of query planning that uses Language Models to translate natural language queries into executable query plans. Different from relational query planners, the resulting query plans can contain complex operators that are able to process arbitrary modalities. As part of this paper, we present a first GPT-4 based prototype called CAESURA and show the general feasibility of this idea on two datasets. Finally, we discuss several ideas to improve the query planning capabilities of today's Language Models.

1 INTRODUCTION
Query planning, the basic process of deriving an executable query plan in response to a user query, has conceptually stayed essentially the same since IBM's System R was introduced in 1974 [14]. In traditional DBMSs, a logical plan is first obtained from parsing a SQL query and then optimized by improving the order of the query's operators. In a second step, each logical operator is mapped to a concrete implementation to obtain a physical plan, which is eventually executed. In the past decades, research has focused primarily on improving the efficiency of query plans by improving various individual aspects (e.g. the cost model [3]).

However, this traditional approach to query planning is fundamentally limited in two important aspects: Firstly, it only applies to query languages such as SQL, whose semantics of queries are clear and can be easily parsed into (at least canonical) query plans to execute the query. Secondly, it is only possible to query structured relational data stored in tables. Due to these limitations and several new trends, we argue that it is finally time to re-think how query planning is done.

Trend 1: Multi-Modal Data. In today's industries, huge amounts of non-relational multi-modal data (e.g. images, documents, sensor data, ...) need to be stored and processed, which has led to solutions that store data outside classical databases. For instance, medical clinics need to store and analyze MRI scans and patient reports along with structured patient metadata. Since these data modalities cannot be stored and queried easily in today's databases, they are usually stored in data lakes, which, however, makes gaining insights from these modalities hard. To gain insights from such multi-modal data lakes, the data needs to be made accessible first, usually by manually constructing complex data processing pipelines.

This paper is published under the Creative Commons Attribution 4.0 International License. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution, provided that the original work to the authors and CIDR 2025, 18th Annual Conference on Innovative Data Systems Research (CIDR '25), January 14-17, 2024, Chamada, USA.

Palimpsest: Optimizing AI-Powered Analytics with Declarative Query Processing

Chunwei Liu¹, Matthew Russo², Michael Cafarella,
Lei Cao¹, Peter Baile Chen, Zui Chen, Michael Franklin¹,
Tim Kraska, Samuel Madden, Rana Shahout, Gerardo Vitagliano

ABSTRACT
A long-standing challenge in AI-powered analytics systems has been the lack of declarative query processing. Existing systems require spending a large amount of runtime and money executing a query, often in a non-interactive manner. In this paper, we present Palimpsest, a declarative query processing system that allows users to interactively explore data and gain insights from it. Palimpsest is designed to be efficient, allowing users to explore data at a scale of 100-125 tokens per second, assuming that a high-quality open source LLM (e.g. GPT-4o mini) model costs \$100 per million tokens. Palimpsest is designed to be efficient, allowing users to explore data at a scale of 100-125 tokens per second, assuming that a high-quality open source LLM (e.g. GPT-4o mini) model costs \$100 per million tokens. Palimpsest is designed to be efficient, allowing users to explore data at a scale of 100-125 tokens per second, assuming that a high-quality open source LLM (e.g. GPT-4o mini) model costs \$100 per million tokens.

1 INTRODUCTION
The performance gap between traditional data processing components and AI-powered components is profound. Naively scaling AI systems to process workloads with thousands or millions of inputs often requires many iterations. Hence, in recent years, Natural Language Interfaces for databases have emerged, which would allow laypersons to query databases using natural language. However, existing approaches typically translate natural language into SQL [28, 29] and are therefore limited to relational data. Another direction are question-answering systems which work on modalities beyond tables [3]. However, question-answering systems only support queries that are much less expressive than what can be done with SQL.

ThalamusDB: Approximate Query Processing on Multi-Modal Data

SAEHAN JO, Cornell University, USA
IMMANUEL TRUMMER, Cornell University, USA

ABSTRACT
We introduce ThalamusDB, a novel approximate query processing system that processes complex SQL queries on multi-modal data. ThalamusDB supports SQL queries integrating natural language predicates on visual, audio, and text data. To answer such queries, ThalamusDB exploits a collection of zero-shot models in combination with relational processing. ThalamusDB utilizes deterministic approximate query processing, harnessing the relative efficiency of relational processing to mitigate the computational demands of machine learning inference. For evaluating a natural language predicate, ThalamusDB requests a small number of labels from users to specify their preferences on the performance objective regarding the three relevant metrics: approximation error, computation time, and labeling overheads. The ThalamusDB query optimizer chooses operators according to user preferences, prioritizing data processing and requested labels to maximize inference efficiency. We evaluate ThalamusDB on several real-world data sets, taken from Craigslist, YouTube, and Netflix, and show that ThalamusDB achieves an average speedup of 35.0x over MindsDB, an exact processing baseline, and outperforms ABAE, a sampling-based method, in 78.9% of cases.

CCS Concepts
• Information systems → Online analytical processing engines; Data model extensions; • Computing → Logic and Phrases; query languages; • Database management systems → Database models

ACM Reference Format
Saeahan Jo, Immanuel Trummer. 2025. ThalamusDB: Approximate Query Processing on Multi-Modal Data. In Proceedings of the ACM SIGMOD Conference on Management of Data (SIGMOD '25), June 2025, 186–200. <https://doi.org/10.1145/3654989>

1 INTRODUCTION
The performance gap between traditional data processing components and AI-powered components is profound. Naively scaling AI systems to process workloads with thousands or millions of inputs often requires many iterations. Hence, in recent years, Natural Language Interfaces for databases have emerged, which would allow laypersons to query databases using natural language. However, existing approaches typically translate natural language into SQL [28, 29] and are therefore limited to relational data. Another direction are question-answering systems which work on modalities beyond tables [3]. However, question-answering systems only support queries that are much less expressive than what can be done with SQL.

Semantic Operators: A Declarative Model for Rich, AI-based Data Processing

Liana Patel¹, Siddharth Jha¹, Melissa Pan², Harshit Gupta¹, Parth Asawa¹,
Carlos Guestrin¹, Matei Zaharia¹

ABSTRACT
The semantic capabilities of large language models (LLMs) have the potential to enable rich analytics and reasoning over vast knowledge corpora. Unfortunately, existing systems either empirically optimize expensive LLM-powered operations with no performance guarantees, or serve a limited set of row-wise LLM operations, providing limited robustness, expressiveness and usability. We introduce semantic operators, the first formalism for declarative and general-purpose AI-based transformations based on natural language specifications (e.g. filtering, sorting, joining or aggregating using natural language criteria). Each operator defines the expected behavior of each operation, similar to relational algebra, and we develop a query optimizer that provides a declarative model for rich, AI-based data processing. Using this approach, we design a group-by semantic operator to accelerate a common application of LLMs: summarizing text. The LOTUS operator effectively synthesizes structured table outputs from unstructured text, similar to what RDBMSes had for tabular data. We show that LOTUS achieves a 170% speedup over existing state-of-the-art LLM-based operators, while maintaining high accuracy. We also show that LOTUS achieves a 170% speedup over existing state-of-the-art LLM-based operators, while maintaining high accuracy. We also show that LOTUS achieves a 170% speedup over existing state-of-the-art LLM-based operators, while maintaining high accuracy.

1 INTRODUCTION
The semantic capabilities of large language models (LLMs) have the potential to enable rich analytics and reasoning over vast knowledge corpora. Unfortunately, existing systems either empirically optimize expensive LLM-powered operations with no performance guarantees, or serve a limited set of row-wise LLM operations, providing limited robustness, expressiveness and usability. We introduce semantic operators, the first formalism for declarative and general-purpose AI-based transformations based on natural language specifications (e.g. filtering, sorting, joining or aggregating using natural language criteria). Each operator defines the expected behavior of each operation, similar to relational algebra, and we develop a query optimizer that provides a declarative model for rich, AI-based data processing. Using this approach, we design a group-by semantic operator to accelerate a common application of LLMs: summarizing text. The LOTUS operator effectively synthesizes structured table outputs from unstructured text, similar to what RDBMSes had for tabular data. We show that LOTUS achieves a 170% speedup over existing state-of-the-art LLM-based operators, while maintaining high accuracy. We also show that LOTUS achieves a 170% speedup over existing state-of-the-art LLM-based operators, while maintaining high accuracy.

SwellIDB: Dynamic Query-Driven Table Generation with Large Language Models

Victor Giannakouris, Immanuel Trummer

ABSTRACT
We present SwellIDB, a data system that enables analytical query processing over tables generated dynamically with Large Language Models (LLMs). Typically, LLMs generate tables on demand under the closed-world assumption, where queries are executed on pre-existing, structured tables. Structured and queryable data offers several key advantages. First, database systems enable users to leverage SQL, a widely adopted and standardized language over the past decades, for executing analytical queries and retrieving precise results. Second, structured data facilitates seamless interaction with external services, such as mobile applications and APIs. However, while querying structured data is highly desirable, the vast majority of data resides in external semi-structured or unstructured sources. Consequently, end-users are often burdened with developing custom Extract-Transform-Load (ETL) workflows to create structured tables, enabling them to query this data using SQL.

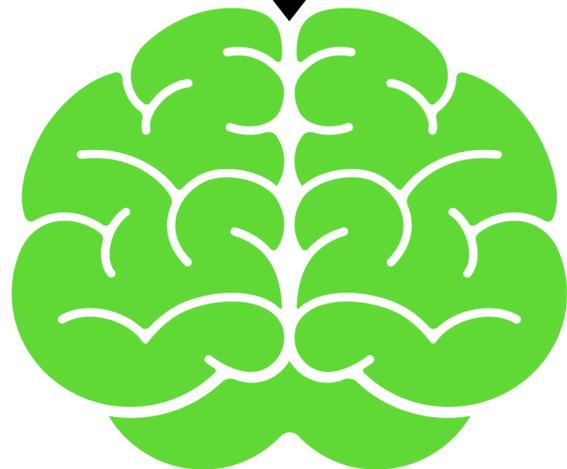
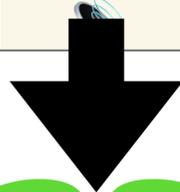
1 INTRODUCTION
We present SwellIDB, a data system that follows a new architecture for analytical query execution over dynamically generated tables. SwellIDB is a data system that follows a new architecture for analytical query execution over dynamically generated tables. SwellIDB is a data system that follows a new architecture for analytical query execution over dynamically generated tables.

Lowering LLM Costs

Lowering LLM Costs

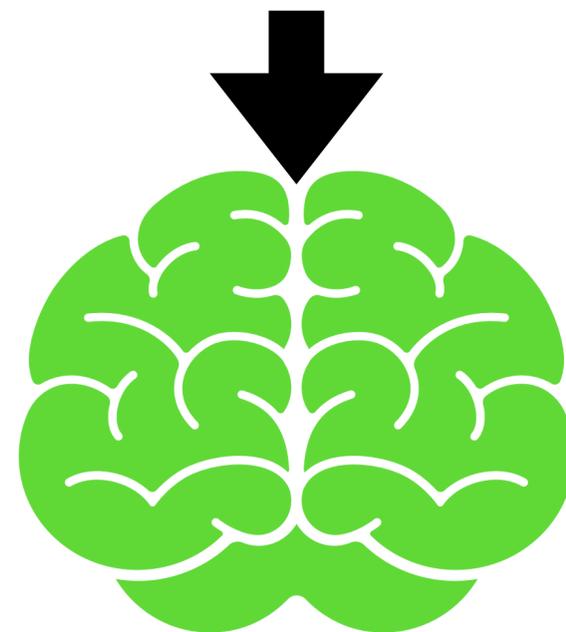
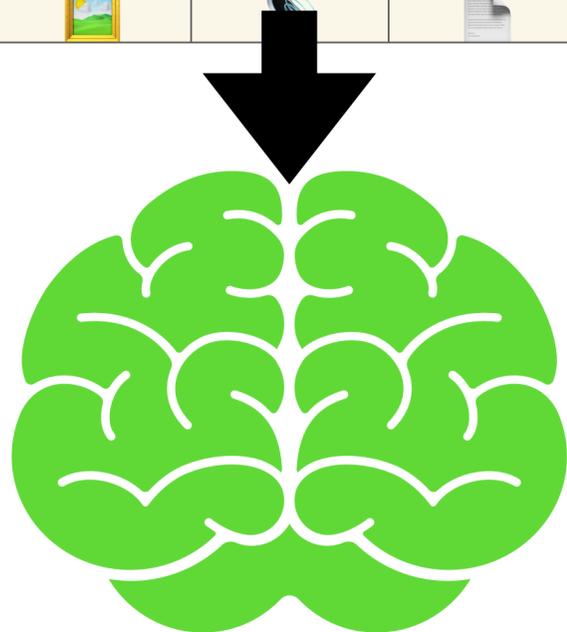


Lowering LLM Costs

Fewer Rows
Approximate Processing



Lowering LLM Costs

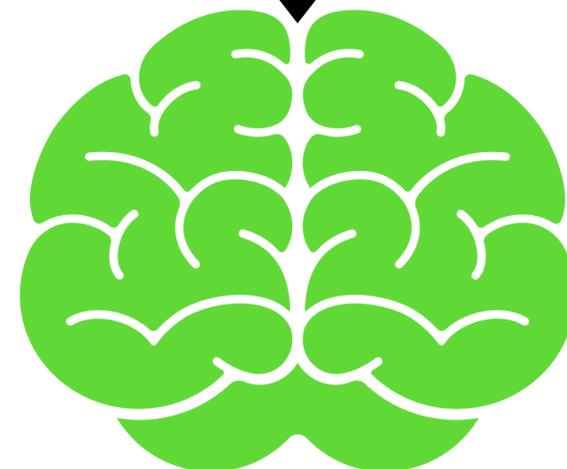
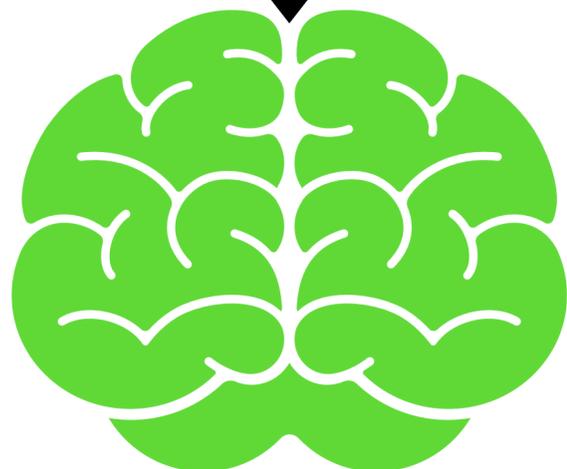
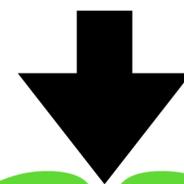
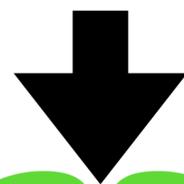
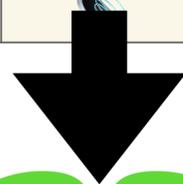
Fewer Rows
Approximate Processing

Fewer Tokens
Data Compression

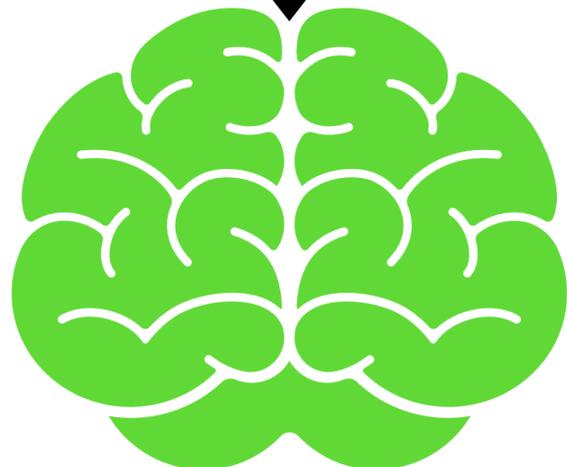
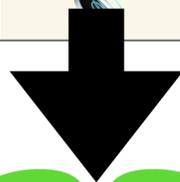
		
		
		
		



Lowering LLM Costs

Fewer Rows
Approximate Processing

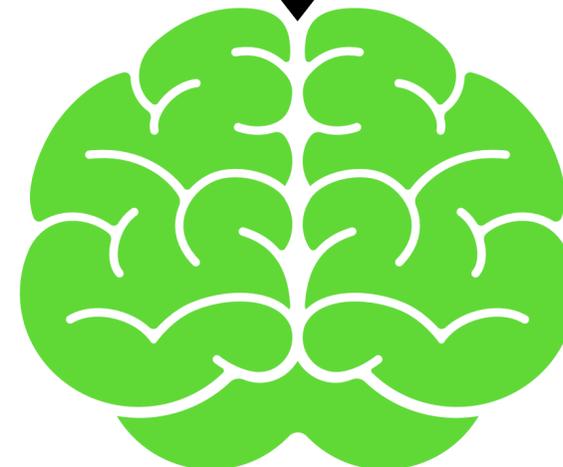
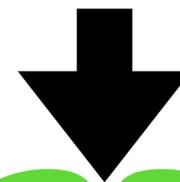
		
		
		
		
		
		
		
		
		



Fewer Tokens
Data Compression

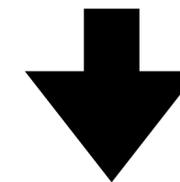
		
		
		
		



Fewer Parameters
Query Optimization



Summary

- Dramatic **expansion of scope** in data processing
 - Data location
 - Data types
 - DB Operators
- Key challenge: **scale it up!**
 - Approximate processing
 - Specialized operators
 - Query optimization

